Hardware/Software Codesign Overview

RASSP Education & Facilitation Program
Module 14

Version 3.00

Copyright © 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute, and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained herein may be duplicated for non-commercial educational use provided this copyright notice is included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds “Unlimited Rights” in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work belong to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this legend.
Rapid Prototyping Design Process

REUSE DESIGN LIBRARIES AND DATABASE

Primarily software

VIRTUAL PROTOTYPE

Primarily hardware

SYSTEM DEF.

FUNCTION DESIGN

HW & SW PART.

HW DESIGN

HW FAB

INTEG.

& TEST

HW & SW CODESIGN

SW DESIGN

SW CODE

HW & SW Partitioning & Codesign
Module Goals

- Introduce the fundamentals of HW/SW codesign and partitioning concepts in designing embedded systems
  - Discuss the current trends in the codesign of embedded systems
  - Provide information on the goals of and methodology for partitioning hardware/software in systems

- Show benefits of the codesign approach over current design process
  - Provide information on how to incorporate these techniques into a general digital design methodology for embedded systems

- Illustrate how codesign concepts are being introduced into design methodologies
  - Several example codesign systems are discussed
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign
- Hardware/Software Codesign Research
- Summary
Module Outline

• Introduction
  • Unified HW/SW Representations
  • HW/SW Partitioning Techniques
  • Integrated HW/SW Modeling Methodologies
  • HW and SW Synthesis Methodologies
  • Industry Approaches to HW/SW Codesign
  • Hardware/Software Codesign Research
  • Summary
Codesign Definition and Key Concepts

● Codesign
  ♦ The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design

● Key concepts
  ♦ Concurrent: hardware and software developed at the same time on parallel paths
  ♦ Integrated: interaction between hardware and software development to produce design meeting performance criteria and functional specs
Motivations for Codesign

- Factors driving codesign (hardware/software systems):
  - Instruction Set Processors (ISPs) available as cores in many design kits (386s, DSPs, microcontrollers, etc.)
  - Systems on Silicon - many transistors available in typical processes (> 10 million transistors available in IBM ASIC process, etc.)
  - Increasing capacity of field programmable devices - some devices even able to be reprogrammed on-the-fly (FPGAs, CPLDs, etc.)
  - Efficient C compilers for embedded processors
  - Hardware synthesis capabilities
Motivations for Codesign (cont.)

- The importance of codesign in designing hardware/software systems:
  - Improves design quality, design cycle time, and cost
    - Reduces integration and test time
  - Supports growing complexity of embedded systems
  - Takes advantage of advances in tools and technologies
    - Processor cores
    - High-level hardware synthesis capabilities
    - ASIC development
Categorizing Hardware/Software Systems

● Application Domain
  ○ Embedded systems
    □ Manufacturing control
    □ Consumer electronics
    □ Vehicles
    □ Telecommunications
    □ Defense Systems
  ○ Instruction Set Architectures
  ○ Reconfigurable Systems

● Degree of programmability
  ○ Access to programming
  ○ Levels of programming

● Implementation Features
  ○ Discrete vs. integrated components
  ○ Fabrication technologies
Categories of Codesign Problems

● Codesign of embedded systems
  ○ Usually consist of sensors, controller, and actuators
  ○ Are reactive systems
  ○ Usually have real-time constraints
  ○ Usually have dependability constraints

● Codesign of ISAs
  ○ Application-specific instruction set processors (ASIPs)
  ○ Compiler and hardware optimization and trade-offs

● Codesign of Reconfigurable Systems
  ○ Systems that can be personalized after manufacture for a specific application
  ○ Reconfiguration can be accomplished before execution or concurrent with execution (called evolvable systems)
Components of the Codesign Problem

- Specification of the system
- **Hardware/Software Partitioning**
  - Architectural assumptions - type of processor, interface style between hardware and software, etc.
  - Partitioning objectives - maximize speedup, latency requirements, minimize size, cost, etc.
  - Partitioning strategies - high level partitioning by hand, automated partitioning using various techniques, etc.
- **Scheduling**
  - Operation scheduling in hardware
  - Instruction scheduling in compilers
  - Process scheduling in operating systems
- **Modeling the hardware/software system during the design process**
Embedded Systems

Application-specific systems which contain hardware and software tailored for a particular task and are generally part of a larger system (e.g., industrial controllers)

- **Characteristics**
  - Are dedicated to a particular application
  - Include processors dedicated to specific functions
  - Represent a subset of reactive (responsive to external inputs) systems
  - Contain real-time constraints
  - Include requirements that span:
    - Performance
    - Reliability
    - Form factor
Embedded Systems: Specific Trends

- Use of microprocessors only one or two generations behind state-of-the-art for desktops
  - E.g. $N/2$ bit width where $N$ is the bit width of current desktop systems
- Contain limited amount of memory
- Must satisfy strict real-time and/or performance constraints
- Must optimize additional design objectives:
  - Cost
  - Reliability
  - Design time
- Increased use of hardware/software codesign principles to meet constraints
Embedded Systems: Examples

- Banking and transaction processing applications
- Automobile engine control units
- Signal processing applications
- Home appliances (microwave ovens)
- Industrial controllers in factories
- Cellular communications
Embedded Systems: Complexity Issues

● Complexity of embedded systems is continually increasing
● Number of states in these systems (especially in the software) is very large
● Description of a system can be complex, making system analysis extremely hard
● Complexity management techniques are necessary to model and analyze these systems
● Systems becoming too complex to achieve accurate “first pass” design using conventional techniques
● New issues rapidly emerging from new implementation technologies
Techniques to Support Complexity Management

- Delayed HW/SW partitioning
  - Postpone as many decisions as possible that place constraints on the design
- Abstractions and decomposition techniques
- Incremental development
  - “Growing” software
  - Requiring top-down design
- Description languages
- Simulation
- Standards
- Design methodology management framework
A Model of the Current Hardware/Software Design Process

DOD-STD-2167A

- **HW Development**
  - **Sys/HW Require. Analysis**
  - **Hardware Require. Analysis**
  - **Prelim. Design**
  - **Detailed Design**
  - **Fabric.**

- **SW Development**
  - **Sys/SW Require. Analysis**
  - **Software Require. Analysis**
  - **Prelim. Design**
  - **Detailed Design**
  - **Coding, Unit test., Integ. test**

**System Integ. and test**

**Operation. Testing and Eval.**

**CSCI Testing**

[Franke91]
Current Hardware/Software Design Process

- **Basic features of current process:**
  - System immediately partitioned into hardware and software components
  - Hardware and software developed separately
  - "Hardware first" approach often adopted

- **Implications of these features:**
  - HW/SW trade-offs restricted
    - Impact of HW and SW on each other cannot be assessed easily
  - Late system integration

- **Consequences these features:**
  - Poor quality designs
  - Costly modifications
  - Schedule slippages
Incorrect Assumptions in Current Hardware/Software Design Process

- 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
Directions of the HW/SW Design Process

Integrated Modeling Substrate

System Concepts
- Sys/HW Require. Analysis
- Sys/SW Require. Analysis

HW Development
- Hardware Require. Analysis
- Prelim. Design
- Detailed Design
- Fabric
- HWCI Testing

SW Development
- Software Require. Analysis
- Prelim. Design
- Detailed Design
- Coding, Unit test., Integ. test
- CSCI Testing
- System Integ. and test
- Operation. Testing and Evaluation

[Franke91]
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 modeling 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
Cross-fertilization Between Hardware and Software Design

- 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
Cross-fertilization Between Hardware and Software Design (cont.)

EDA tool technology has been transferred to SW CAD systems

- Designer support (not automation)
- Graphics-driven design
- Central database for design information
- Tools to check design behavior early in process
Cross-fertilization Between Hardware and Software Design (cont.)

- **Software technology has been transferred to EDA tools**
  - Single-language design
    - Use of 1 common language for architecture spec. and implementation of a chip
  - Compiler-like transformations and techniques
    - Dead code elimination
    - Loop unrolling
  - Design change management
    - Information hiding
    - Design families
Typical Codesign Process

- System Description (Functional)
- Concurrent processes
- Programming languages
- HW/SW Partitioning
- Unified representation (Data/control flow)
- Software Synthesis
- Interface Synthesis
- Hardware Synthesis
- System Integration
- Instruction set level
- HW/SW evaluation

FSM-directed graphs

Another HW/SW partition
Codesign Features

Basic features of a codesign process

- Enables mutual influence of both HW and SW early in the design cycle
  - Provides continual verification throughout the design cycle
  - Separate HW/SW development paths can lead to costly modifications and schedule slippages

- Enables evaluation of larger design space through tool interoperability and automation of codesign at abstract design levels

- Advances in key enabling technologies (e.g., logic synthesis and formal methods) make it easier to explore design tradeoffs
State of Codesign Technology

● Current use limited by:
  ○ Lack of a standardized representation
  ○ Lack of good validation and evaluation methods

● Possible solutions:
  ○ Extend existing hardware/software languages to the use of heterogeneous paradigms
  ○ Extend formal verification techniques to the HW/SW domain
Issues and Problems: Integration

- Errors in hardware and software design become much more costly as more commitments are made.
- “Hardware first” approach often compounds software cost because software must compensate for hardware inadequacies.

![Graph showing software cost impact of inadequate hardware resources.](Image)
Module Outline

- Introduction

**Unified HW/SW Representations**

- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign
- Hardware/Software Codesign Research
- Summary
Unified HW/SW Representation

- **Unified Representation** --
  - A representation of a system that can be used to describe its functionality independent of its implementation in hardware or software
  - Allows hardware/software partitioning to be delayed until trade-offs can be made
  - Typically used at a high-level in the design process

- Provides a simulation environment after partitioning is done, for both hardware and software designers to use to communicate

- Supports cross-fertilization between hardware and software domains
Current Abstraction Mechanisms in Hardware Systems

Abstraction
The level of detail contained within the system model

- A system can be modeled at system, instruction set, register-transfer, logic, or circuit level

- A model can describe a system in the behavioral, structural, or physical domain
<table>
<thead>
<tr>
<th>Level</th>
<th>Behavior</th>
<th>Structure</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMS (System)</td>
<td>Communicating Processes</td>
<td>Processors</td>
<td>Cabinets, Cables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memories</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switches (PMS)</td>
<td></td>
</tr>
<tr>
<td>Instruction Set</td>
<td>Input-Output</td>
<td>Memory, Ports</td>
<td>Board</td>
</tr>
<tr>
<td>(Algorithm)</td>
<td></td>
<td>Processors</td>
<td>Floorplan</td>
</tr>
<tr>
<td>Register-Transfer</td>
<td>Register Transfers</td>
<td>ALUs, Regs, Muxes, Bus</td>
<td>ICs</td>
</tr>
<tr>
<td>Logic</td>
<td>Logic Equns.</td>
<td>Gates, Flip-flops</td>
<td>Std. cell layout</td>
</tr>
<tr>
<td>Circuit</td>
<td>Network Equns.</td>
<td>Trans., Connections</td>
<td>Transistor layout</td>
</tr>
</tbody>
</table>
Virtual machine

A software layer very close to the hardware that hides the hardware’s details and provides an abstract and portable view to the application programmer

Attributes

- Developer can treat it as the real machine
- A convenient set of instructions can be used by developer to model system
- Certain design decisions are hidden from the programmer
- Operating systems are often viewed as virtual machines
Virtual Machine Hierarchy

- Application Programs
- Utility Programs
- Operating System
- Monitor
- Machine Language
- Microcode
- Logic Devices
Abstract Hardware-Software Model

Uses a unified representation of system to allow early performance analysis

- General Performance Evaluation
- Identification of Bottlenecks
- Evaluation of HW/SW Model
- Evaluation of Design Alternatives
- Evaluation of HW/SW Trade-offs
Examples of Unified HW/SW Representations

Systems can be modeled at a high level as:

- Data/control flow diagrams
- Concurrent processes
- Finite state machines
- Object-oriented representations
- Petri Nets
● Data/control flow graphs
  ○ Graphs contain nodes corresponding to operations in either hardware or software
  ○ Often used in high-level hardware synthesis
  ○ Can easily model data flow, control steps, and concurrent operations because of its graphical nature

Example:

```
  5  X  4  Y
/   |   |
|   +   |
|   |   |
|   +   |
  ———— + ————
|   |   |
|   +   |
  ———— + ————
|   |   |
|   +   |

Control Step 1
Control Step 2
Control Step 3
```
Unified Representations (Cont.)

- Concurrent processes
  - Interactive processes executing concurrently with other processes in the system-level specification
  - Enable hardware and software modeling

- Finite state machines
  - Provide a mathematical foundation for verifying system correctness, simulation, hardware/software partitioning, and synthesis
  - Multiple FSMs that communicate can be used to model reactive real-time systems
Unified Representations (Cont.)

- **Object-oriented representations:**
  - Use techniques previously applied to software to manage complexity and change in hardware modeling
  - Use C++ to describe hardware and display OO characteristics
  - Use OO concepts such as
    - Data abstraction
    - Information hiding
    - Inheritance
  - Use building block approach to gain OO benefits
    - Higher component reuse
    - Lower design cost
    - Faster system design process
    - Increased reliability
Unified Representations (Cont.)

Object-oriented representation

Example:

3 Levels of abstraction:

- Register
  - Read
  - Write

- ALU
  - Add
  - Sub
  - AND
  - Shift

- Processor
  - Mult
  - Div
  - Load
  - Store
Unified Representations (Cont.)

- Petri Nets: a system model consisting of places, tokens, transitions, arcs, and a marking
  - Places - equivalent to conditions and hold tokens
  - Tokens - represent information flow through system
  - Transitions - associated with events, a “firing” of a transition indicates that some event has occurred
  - Marking - a particular placement of tokens within places of a Petri net, representing the state of the net

Example:

```
Input Places
          ∙

Transition

Token

Output Place
          ∙
```
Module Outline

- Introduction
- Unified HW/SW Representations

**HW/SW Partitioning Techniques**
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign
- Hardware/Software Codesign Research
- Summary
Hardware/Software Partitioning

● 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.)

● This is a multivariate optimization problem that when automated, is an NP-hard problem
HW/SW Partitioning Issues

- Partitioning into hardware and software affects overall system cost and performance

  - **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
Partitioning Approaches

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

- Start with all functionality in hardware and move portions into software implementation (hardware-oriented partitioning)
System Partitioning (Functional Partitioning)

- System partitioning in the context of hardware/software codesign is also referred to as *functional partitioning*.

- Partitioning functional objects among system components is done as follows:
  - The system’s functionality is described as collection of indivisible functional objects.
  - Each system component’s functionality is implemented in either hardware or software.

- An important advantage of functional partitioning is that it allows hardware/software solutions.
Partitioning Metrics

- **Deterministic estimation techniques**
  - Can be used only with a fully specified model with all data dependencies removed and all component costs known
  - Result in very good partitions

- **Statistical estimation techniques**
  - Used when the model is not fully specified
  - Based on the analysis of similar systems and certain design parameters

- **Profiling techniques**
  - Examine control flow and data flow within an architecture to determine computationally expensive parts which are better realized in hardware
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
Hardware/Software System Architecture Trends

- Some operations in special-purpose hardware
  - Generally take the form of a coprocessor communicating with the CPU over its bus
    - Computation must be long enough to compensate for the communication overhead
  - May be implemented totally in hardware to avoid instruction interpretation overhead
    - Utilize high-level synthesis algorithms to generate a register transfer implementation from a behavior description

- Partitioning algorithms are closely related to the process scheduling model used for the software side of the implementation
A hardware/software partition is defined using two sets $H$ and $S$, where $H \subset O$, $S \subset O$, $H \cup S = O$, $H \cap S = \emptyset$.

**Associated metrics:**
- $\text{Hsize}(H)$ is the size of the hardware needed to implement the functions in $H$ (e.g., number of transistors).
- $\text{Performance}(G)$ is the total execution time for the group of functions in $G$ for a given partition $\{H, S\}$.
- Set of performance constraints, $Cons = (C_1, \ldots, C_m)$, where $C_j = \{G, timecon\}$, indicates the maximum execution time allowed for all the functions in group $G$ and $G \subset O$. 
A performance satisfying partition is one for which performance($C_j.G$) $\leq C_j.timecon$, for all $j=1...m$

Given $O$ and $Cons$, the hardware/software partitioning problem is to find a performance satisfying partition $\{H,S\}$ such that $Hsize(H)$ is minimized

The all-hardware size of $O$ is defined as the size of an all hardware partition (i.e., $Hsize(O)$)
Issues in Partitioning

- Specification abstraction level
- Granularity
- System-component allocation
- Metrics and estimations
- Partitioning algorithms
- Objective and closeness functions
- Partitioning algorithms
- Output
- Flow of control and designer interaction
Issues in Partitioning (Cont.)

High Level Abstraction

Decomposition of functional objects

- Metrics and estimations
- Partitioning algorithms
- Objective and closeness functions

Component allocation

Output
Specification Abstraction Levels

- **Task-level dataflow graph**
  - A Dataflow graph where each operation represents a task

- **Task**
  - Each task is described as a sequential program

- **Arithmetic-level dataflow graph**
  - A Dataflow graph of arithmetic operations along with some control operations
  - The most common model used in the partitioning techniques

- **Finite state machine (FSM) with datapath**
  - A finite state machine, with possibly complex expressions being computed in a state or during a transition
● Register transfers
  ○ The transfers between registers for each machine state are described

● Structure
  ○ A structural interconnection of physical components
  ○ Often called a netlist
Granularity Issues in Partitioning

- The granularity of the decomposition is a measure of the size of the specification in each object

- The specification is first decomposed into functional objects, which are then partitioned among system components
  - Coarse granularity means that each object contains a large amount of the specification.
  - Fine granularity means that each object contains only a small amount of the specification
    - Many more objects
    - More possible partitions
      - Better optimizations can be achieved
System Component Allocation

- The process of choosing system component types from among those allowed, and selecting a number of each to use in a given design
- The set of selected components is called an allocation
  - Various allocations can be used to implement a specification, each differing primarily in monetary cost and performance
  - Allocation is typically done manually or in conjunction with a partitioning algorithm
- A partitioning technique must designate the types of system components to which functional objects can be mapped
  - ASICs, memories, etc.
Metrics and Estimations

Issues

● A technique must define the attributes of a partition that determine its quality
  ○ Such attributes are called *metrics*
    □ Examples include monetary cost, execution time, communication bit-rates, power consumption, area, pins, testability, reliability, program size, data size, and memory size
    □ *Closeness metrics* are used to predict the benefit of grouping any two objects

● Need to compute a metric’s value
  ○ Because all metrics are defined in terms of the structure (or software) that implements the functional objects, it is difficult to compute costs as no such implementation exists during partitioning
Two key metrics are used in hardware/software partitioning

- **Performance**: Generally improved by moving objects to hardware

- **Hardware size**: Hardware size is generally improved by moving objects out of hardware
Computation of Metrics

Two approaches to computing metrics

- Creating a detailed implementation
  - Produces accurate metric values
  - Impractical as it requires too much time
- Creating a rough implementation
  - Includes the major register transfer components of a design
  - Skips details such as precise routing or optimized logic, which require much design time
  - Determining metric values from a rough implementation is called estimation
Objective and Closeness Functions

- Multiple metrics, such as cost, power, and performance are weighed against one another
  - An expression combining multiple metric values into a single value that defines the quality of a partition is called an *Objective Function*
  - The value returned by such a function is called *cost*
  - Because many metrics may be of varying importance, a weighted sum objective function is used
    - e.g., $\text{Objfct} = k_1 \times \text{area} + k_2 \times \text{delay} + k_3 \times \text{power}$
  - Because constraints always exist on each design, they must be taken into account
    - e.g $\text{Objfct} = k_1 \times F(\text{area, area\_constr})$
      + $k_2 \times F(\text{delay, delay\_constr})$
      + $k_3 \times F(\text{power, power\_constr})$
Partitioning Algorithm Issues

- Given a set of functional objects and a set of system components, a partitioning algorithm searches for the best partition, which is the one with the lowest cost, as computed by an objective function.

- While the best partition can be found through exhaustive search, this method is impractical because of the inordinate amount of computation and time required.

- The essence of a partitioning algorithm is the manner in which it chooses the subset of all possible partitions to examine.
Partitioning Algorithm Classes

- **Constructive algorithms**
  - Group objects into a complete partition
  - Use closeness metrics to group objects, hoping for a good partition
  - Spend computation time constructing a small number of partitions

- **Iterative algorithms**
  - Modify a complete partition in the hope that such modifications will improve the partition
  - Use an objective function to evaluate each partition
  - Yield more accurate evaluations than closeness functions used by constructive algorithms

- **In practice, a combination of constructive and iterative algorithms is often employed**
Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent evaluating large numbers of partitions.
- Iterative algorithms differ from one another primarily in the ways in which they modify the partition and in which they accept or reject bad modifications.
- The goal is to find global minimum while performing as little computation as possible.

A, B – Local minima
C – Global minimum
Iterative Partitioning Algorithms (Cont.)

- Two broad categories:
  - Greedy algorithms
    - Only accept moves that decrease cost
    - Can get trapped in local minima
  - Hill-climbing algorithms
    - Allow moves in directions increasing cost (retracing)
      - Through use of stochastic functions
    - Can escape local minima
    - E.g., simulated annealing
Any partitioning technique must define the representation format and potential use of its output

- E.g., the format may be a list indicating which functional object is mapped to which system component
- E.g., the output may be a revised specification
  - Containing structural objects for the system components
  - Defining a component’s functionality using the functional objects mapped to it
Flow of Control and Designer Interaction

- Sequence in making decisions is variable, and any partitioning technique must specify the appropriate sequences
  - E.g., selection of granularity, closeness metrics, closeness functions

- Two classes of interaction
  - Directives
    - Include possible actions the designer can perform manually, such as allocation, overriding estimations, etc.
  - Feedback
    - Describe the current design information available to the designer (e.g., graphs of wires between objects, histograms, etc.)
Comparing Partitions Using Cost Functions

- A cost function is a function $\text{Cost}(H, S, Cons, I)$ which returns a natural number that summarizes the overall quality of a given partition
  - $I$ contains any additional information that is not contained in $H$ or $S$ or $Cons$
  - A smaller cost function value is desired

- An iterative improvement partitioning algorithm is defined as a procedure
  \[
  \text{Part}_\text{Alg}(H, S, Cons, I, \text{Cost}( ))
  \]
  which returns a partition $H', S'$ such that
  \[
  \text{Cost}(H', S', Cons, I) \leq \text{Cost}(H, S, Cons, I)
  \]
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques

**Integrated HW/SW Modeling Methodologies**

- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign
- Hardware/Software Codesign Research
- Summary
Cosimulation

- An HDL (VHDL or Verilog) simulation environment is used to perform behavioral simulation of the system hardware processes.

- A Software environment (C or C++) is used to develop the code.

- SW and HW execute as separate processes linked through UNIX IPC (interprocessor communications) mechanisms (sockets).
Verilog Cosimulation Example

Software processes communicate with hardware simulator via UNIX sockets.

Verilog PLI (programming language interface) serves as translator, allowing hardware simulation models to communicate with software processes.

Verilog HW Simulator

Module: Application specific hardware

HW proc 1

HW proc 2

Module: Bus Interface

Verilog PLI

SW proc 1

UNIX sockets

SW proc 2

[Thomas93]
VHDL Cosimulation Example

Software processes communicate with hardware simulator via foreign language interface

Allowing hardware simulation models to “cosimulate” with software processes.
VHDL-C Based HW/SW Cosimulation for DSP Multicomputer Application

Algorithm - C

Scheduler - C

Mapping Function (e.g.):
- Round Robin
- Computational Requirements Based
- Communications Requirements Based

Architecture - VHDL

CPU 1  CPU 2  CPU 3  CPU 4

Communications Network
Unix C Program

System State (e.g.):

**CPU:**
Time to instruction completion

**Comm Agent:**
Messages in Send Queue
Messages in Recv Queue

**Network:**
Communications Channels Busy

Next Instruction for CPU to Execute (e.g.):

Send(destination, message_length)
Recv(source, message_length)
Compute(time)

VHDL Simulator

Architecture Model

**INSTRUMENT**

**PACKAGE**

**CPU 1**  **CPU 2**  **CPU 3**  **CPU 4**

**Comm Agent 1**  **Comm Agent 2**  **Comm Agent 3**  **Comm Agent 4**

**Communications Network**
Model Continuity Problem

Inability to gradually define a system-level model into a hardware/software implementation

- Model continuity problems exist in both hardware and software systems
- Model continuity can help address several system design problems
  - Allows validation of system level models with corresponding HW/SW implementation
  - Addresses subsystem integration
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- **HW and SW Synthesis Methodologies**
  - Industry Approaches to HW/SW Codesign
  - Hardware/Software Codesign Research
- Summary
Hardware Design Methodology

Hardware Design Process:
Waterfall Model

- Hardware Requirements
- Preliminary Hardware Design
- Detailed Hardware Design
- Fabrication
- Testing

Copyright © 1995-1999 SCRA
Hardware Design Methodology (Cont.)

- Use of HDLs for modeling and simulation
- Use of lower-level synthesis tools to derive register transfer and lower-level designs
- Use of high-level hardware synthesis tools
  - Behavioral descriptions
  - System design constraints
- Introduction of synthesis for testability at all levels
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 behavior 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)
Software Design Methodology

Software Design Process:
Waterfall Model

1. Software Requirements
2. Software Design
3. Coding
4. Testing
5. Maintenance

Copyright © 1995-1999 SCRA
Software Design Methodology (Cont.)

- Software requirements includes both
  - Analysis
  - Specification
- Design: 2 levels:
  - System level - module specs.
  - Detailed level - process design language (PDL) used
- Coding - in high-level language
  - C/C++
- Maintenance - several levels
  - Unit testing
  - Integration testing
  - System testing
  - Regression testing
  - Acceptance testing
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
Why Use 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
Software Synthesis Categories

- **Language compilers**
  - ADA and C compilers
  - YACC - yet another compiler compiler
  - Visual Basic

- **Domain-specific synthesis**
  - Application generators from software libraries
Software Synthesis Examples

- **Mentor Graphics Concurrent Design Environment System**
  - Uses object-oriented programming (written in C++)
  - Allows communication between hardware and software synthesis tools

- **Index Technologies Excelerator and Cadre’s Teamwork Toolsets**
  - Provide an interface with COBOL and PL/1 code generators

- **KnowledgeWare’s IEW Gamma**
  - Used in MIS applications
  - Can generate COBOL source code for system designers

- **MCCI’s Graph Translation Tool (GrTT)**
  - Used by Lockheed Martin ATL
  - Can generate ADA from Processing Graph Method (PGM) graphs
GrTT Tool Architecture

Methodology
Reinventing
Electronic
Design
Architecture
Infrastructure

*Signal Processing Graph Notation

Constraints/Error Cond.
Behavior
Code Fragments

SPGN*
PARCER

GRAPH
ANALYSIS

AUTOCODER

SPGN File
GV Sets

Validated Graph Object
Behavioral Specification

Ada Source Code File

Domain Primitive Database

Copyright © 1995-1999 SCRA
Interface Synthesis

- **Definition**: the automatic design and implementation of hardware (glue logic) and the software (drivers) components between the processor and the dedicated hardware.

- **Goals/benefits**
  - To quickly create and modify designs
  - To remove from the designer the handling of the tedious details of VLSI design
Typical approaches use standard interface schemes

- memory-mapped
- serial port
- parallel port
- self-timed
- synchronous
- blocking
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
Cosynthesis Approach to System Implementation

- Memory
- Behavioral Specification and Performance criteria
- System Input
- System Output

Performance

Cost

Mixed Implementation

Pure HW

Pure SW

Constraints

System Input

© IEEE 1993

[Gupta93]
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies

**Industry Approaches to HW/SW Codesign**
- Hardware/Software Codesign Research
- Summary
Sanders Codesign Methodology

Global influences
- Libraries
- Design rules
- Tool select.
- Virtual Environ.
- Cost models

Feedback to user
- At all steps

Requirements
- Req. Analysis
- Algorithm Develop.
- HW/SW Tradeoff Analysis

Integrate & Test

System Checkout

Hardware Modules
- Logical & Phys. Design
- Anal. & Simul.

Software Modules
- Design
- Code
- Test
- Integrated HW/SW Simulation

Design
- SW Req. Partition.

Development
- HW Req. Partition.
Sanders Codesign Methodology
Integrated Modeling Substrate

- System Requirements
- Arch Ind. Proc Model

Hardware Perf. Model
Behavior Level Model
ISA Model
RTL Model
Gate Level Model
Prototype Hardware

Simulation Library

Software Perf. Model
Arch Dep. Proc Model
Source Code
HOL
Assembly
Load Module

Copyright © 1995-1999 SCRA

[RASSP94]
Subsystems process
- Processing requirements are modeled in an architecture-independent manner
- Codesign not an issue

Architecture process
- HW/SW allocation analyzed via modeling of SW performance on candidate architectures
- Hierarchical verification is performed using finer grain modeling (ISA and below)

Detailed design
- Downloadable executable application and test code is verified to maximum extent possible

Library support
- SW models validated on test data
- HW models validated using existing SW models
- HW & SW models jointed iterated throughout designs
Lockheed Martin ATL Codesign Methodology
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign

**Hardware/Software Codesign Research**

- Summary
Major Codesign Research Efforts

- **Chinook - University of Washington** - Chou, Ortega, Borriello
- **Cosmos - Grenoble University** - Ismail, Jerraya
- **Cosyma - University of Braunschweig** - Ernst, Henkel, Benner
- **Polis - U. C. Berkeley** - Chiodo, Giusto, Jurecska, Hsieh, Lavagno, Sangiovanni-Vincentelli
- **Ptolemy - U. C. Berkeley** - Kalavade, Lee
- **Siera- U. C. Berkeley** - Srivastava, Broderson
Chinook

- Unified representation: Event Graph (CDFG)
- Partitioning: constraint driven by scheduling requirements
- Scheduling: timing driven
- Modeling substrate: based on Verilog HDL
- Validation: simulation based (Verilog)
- Main emphasis on synthesis of hardware/software interfaces
**Cosmos**

- Unified representation: Initial description is done in SDL (specification description language) which is translated into SOLAR, an intermediate form that allows several description levels (CSPs, FSMs, etc.)
- Partitioning: user driven using a tool that allows processes to be grouped together or split into sub-processes
- Scheduling: based on the partitioning
- Modeling substrate: VHDL simulation after architecture mapping
- Validation: simulation based
- Main emphasis on synthesis of communications mechanisms between processes - reuse of existing communication models
Cosyma

- Unified representation: ES graph (CDFG)
- Partitioning: combined method based on course partitioning by user with cost guidance and finer scheduling done by simulated annealing
- Scheduling: no specific method
- Modeling substrate: based on C++
- Validation: simulation based (C++)
- Main emphasis on partitioning for hardware accelerators
Polis

- Unified representation: Codesign Finite State Machine (CFSM) based
- Partitioning: user driven with cost estimated provided by co-simulation
- Scheduling: classical real-time algorithms
- Modeling substrate: Ptolemy based (C++)
- Validation: co-simulation and formal FSM verification
- Main emphasis on verifiable specification not biased to either hardware or software implementation
Ptolemy

- Unified representation: Data Flow Graph
- Partitioning: greedy algorithm based on scheduling constraints
- Scheduling: linear based on sorting blocks by “criticality”
- Modeling substrate: heterogeneous modeling and simulation framework based on C++
- Validation: based on simulation
- Main emphasis on heterogeneous modeling framework (mixing different models of computation)
Siera

- Unified representation: static, hierarchical network of concurrent sequential processes communicating via message queues (similar to DFG)
- Partitioning: manual user driven
- Scheduling: static process to processor mapping, priority based preemptive schedulers available within real-time OS on processors
- Modeling substrate: based on VHDL - includes support for modeling continuous time systems such as sensors and actuators
- Validation: based on simulation
- Main emphasis on the design of embedded systems targeted towards a predefined architectural template
Chinook

- Hardware/Software Co-synthesis system developed at the University of Washington
- Targeted at real-time reactive embedded systems
- Control dominated designs constructed from off-the-shelf components
Chinook’s Principal Innovations

- Single Specification - one specification, with explicit timing/performance constraints is used for the system’s hardware and software
- One Simulation Environment - the high level specification, the final result, and any intermediate steps can be simulated to verify and debug the design
- Software Scheduling - the appropriate software architecture is synthesized to meet the timing requirements
- Interface Synthesis - the hardware and software necessary to interface between system components (glue logic and device drivers) is automatically synthesized
- Complete Information for Physical Prototyping - a complete netlist is generated for the hardware, and C source code is generated for the software
The Chinook System

Verilog Specification

parser

scheduler

comm. synthesizer

code generator

program

Driver synthesizer

interface synthesizer

netlist

Behavioral Simulation

Mixed Simulation

Structural Simulation

Copyright © 1995-1999 SCRA
The system specification is written in a dialect of Verilog and includes the system’s behavior and the structure of the system architecture.

The behavior is specified as a set of tasks in a style similar to communicating finite state machines - control states of the system are organized as *modes* which are behavioral regimes similar to hierarchical states.

In a given mode, the system’s responses are defined by a set of *handlers* which are essentially event-triggered routines.

The designer must *tag* tasks or modules with the processor that is preferred for their implementation - untagged tasks are implemented in software.

The designer can specify response times and rate constraints for tasks in the input description.
Scheduling in Chinook

- Chinook provides an automated scheduling algorithm
- Low-level I/O routines and high level routines grouped in modes are scheduled statically
- A static, nonpreemptive scheduling algorithm is used to meet min/max timing constraints on low-level operations
  - Determines serial ordering for operations
  - Inserts delays as necessary to meet minimum constraints
  - Includes heuristics in the scheduling algorithm to help exact algorithm generate valid solution to NP-hard scheduling problem
- A customized dynamic scheduler may be generated for the top-level modes
Interface Synthesis in Chinook

- Realization of communication between system components is an area of emphasis in the Chinook system
- Chinook synthesizes device drivers from timing diagrams
- Custom code for the processor being used is generated
  - For processors with I/O ports, an efficient heuristic is used to connect devices with minimal interface hardware
  - For processors w/o I/O ports, a memory mapped I/O interface is generated including allocating address spaces, and generating the required bus logic and instructions
- Portions of the interface that cannot be implemented in software are synthesized into external hardware
Communications Synthesis and System Simulation in Chinook

- Chinook provides methods for synthesizing communications systems between multiple processors if a multicomputer implementation is chosen
  - Bus-based, point-to-point, and hybrid communications schemes are supported
  - Communications library that includes FIFOs, arbiters, and interconnect templates is provided

- Simulation of the design at different levels of detail is supported
  - Verilog-XL Programming Language is used
  - Verilog PLI is used to interface to device models written in C
  - Each device supports the same API for simulation and synthesis - API calls can be used by the designer to animate the model interactively
  - RTL level models of the processors are used to simulate the final implementation of the system (software)
Developed at the Technical University of Braunschweig, Germany

An experimental system for HW/SW codesign of small embedded real time systems
- Implements as many operations as possible in software running on a processor core
- Generates external hardware only when timing constraints are violated

Target architecture:
- Standard RISC processor core
- Application-specific processor

Communication between HW and SW through shared memory with a communicating sequential processes (CSP) type protocol
COSYMA (Cont.)

- Input description of system in C* is translated into an internal graph representation supporting:
  - Partitioning
  - Generating hardware descriptions for parts moved to hardware

- Internal graph representation combines:
  - Control and dataflow graph
  - Extended syntax (ES) graph
    - Syntax graph
    - Symbol table
    - Local data/control dependencies
Design Flow in a COSYMA System

C* Mode

C* Compiler

ES Flowgraph

Partitioning

Cost Estimation

Run time Analysis

Simulator

ES to HW C

HW-C Model

Olympus

ES to C

C Program

C Compiler

Object Code
COSYMA - Aims and Strategies

- Major aim is automating HW/SW partitioning process, for which very few tools currently exist

- COSYMA partitions at the basic block and function level (including hierarchical function calls)
  - Simulated annealing algorithm is used because of its flexibility in the cost function and the possibility to trade-off computation time vs result quality
  - Starts with an unfeasible all-software solution
The cost function is defined to force the annealing to reach a feasible solution before other optimization goals (e.g., area).

The metrics used in cost computation are:
- Expected hardware execution times
- Software execution times
- Communication
- Hardware costs

The cost function is updated in each step of the simulated annealing algorithm.
After partitioning, the parts selected to be realized in software are translated to a C program, thereby inserting code for communicating with the coprocessor.

The rest of the system is translated to the input description of the high-level synthesis system, and an application-specific coprocessor is synthesized.

Lastly, a fast-timing analysis of the whole HW/SW system is performed to test whether all constraints are satisfied.
Ptolemy

- A software environment for simulation and prototyping of heterogeneous systems

- Attributes
  - Facilitates mixed-mode system simulation, specification, and design
  - Supports generation of DSP assembly code from a block diagram description of algorithm
  - Uses object-oriented representations to model subsystems efficiently
  - Supports different design styles called domains
Ptolemy supports a framework for hardware/software codesign, called the Design Assistant.

The Design Assistant consists of two components:

- Specific point tools for estimation, partitioning, synthesis, and simulation
- An underlying design methodology management infrastructure for design space exploration
Codesign Methodology Using Ptolemy (Cont.)

Design Flow

- Design constraints
- User inputs
- Design specs.

Area/Time Estimation

HW/SW Partitioning

- Hardware Synthesis
- Interface Synthesis
- Software Synthesis

Netlist Generation

- VHDL/Synopsys
- Ptolemy

System

Layout + Software

Manual
- CPLEX(ILP)
- GCLP...

Simulation

Codesign Methodology
Using Ptolemy (Cont.)

[Rozenblit94]

Design constraints

Design specs.

User inputs

Area/Time Estimation

HW/SW Partitioning

Hardware Synthesis

Interface Synthesis

Software Synthesis

Netlist Generation

VHDL/Synopsys

Ptolemy

System

Layout + Software

Copyright © 1995-1999 SCRA

© IEEE 1994
Data encapsulated in “particles”
“Block” objects send and receive messages
Particles travel to/from external world through “portholes”
POLIS

- Hardware/Software Codesign and synthesis system developed at the University of California, Berkeley

- Targeted towards small, scale, reactive, control dominated embedded systems

- Includes an “unbiased” mechanism for specifying the system’s function that allows for maximum flexibility in mapping to hardware or software and also allows for formal verification
● System behavior is specified in a formal manner using Codesign Finite State Machines (CFSMs)
  ○ CFSMs translate a set of inputs to a set of outputs with only a finite amount of internal state
  ○ Unlike traditional FSMs, CFSMs do not all change state exactly at the same time (globally asynchronous)
● CFSMs are designed to be unbiased towards hardware or software
● Translators exist to convert other specification languages (e.g. ESTEREL) into CFSMs
● CFSMs can be translated into traditional FSMs to allow formal verification
● CFSMs can communicate with each other using events
  ○ Events are unidirectional and happen in non-zero, unbounded time
  ○ Events can be used to communicate across all domains (hardware or software)
  ○ Events are unbuffered and can be overwritten - however, they can be used to implement fully interlocked handshaking
● CFSMs are translated into behavioral FSMs for hardware synthesis and into S-graphs for software synthesis
● Specification: “Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for ten seconds or until the key is turned off”

(*Key == On) → *Start

(*Key == Off) or (*Belt == On) →

(*End == 5) → *Alarm = On

(*End == 10) or (*Belt == On) or (*Key == Off) → *Alarm = Off
Partitioning and Scheduling in POLIS

- Partitioning based on mapping CFSMs to either hardware or software
- This mapping is left to the user - performance feedback is provided by simulation
- Interfaces between partitions are automatically generated
- Scheduling based on executing CFSMs
- Selection of scheduling algorithm left to user - built into RTOS
  - Round-robin cyclic executive
  - Off-line I/O rate-based cyclic executive
  - Static pre-emptive: rate monotonic scheduling
  - Dynamic pre-emptive: Earliest Deadline First
Interfaces Among Partitions

- Interfaces use strobe/data protocol (corresponding to the event/value primitive)

![Diagram of interfaces among partitions]

- Example HW to SW interface

![Example HW to SW interface diagram]
The POLIS Co-design Environment

- Graphical EFSM
- ESTEREL
- (Other)...

Formal Verification

Partitioning

Simulation

SW Synthesis

SW Code + RTOS

Interface Synthesis

Logic Netlist

HW Synthesis

Prototype

Compilers

CFSMs

Copyright © 1995-1999 SCRA
Module Outline

- Introduction
- Unified HW/SW Representations
- HW/SW Partitioning Techniques
- Integrated HW/SW Modeling Methodologies
- HW and SW Synthesis Methodologies
- Industry Approaches to HW/SW Codesign
- Hardware/Software Codesign Research

- Summary
Module Summary

- The synergistic design of hardware and software in a digital system, called *Hardware/Software Codesign*, has been explored.
- Elements of a HW/SW Codesign methodology have been outlined.
- Industrial design flows that contain aspects of codesign have been presented.
- Present day research into automating portions of the codesign problem have been explored.
- As digital systems become more complex and performance criteria become more stringent, codesign will become a necessity.
- Better design tools and unified design environments will allow codesign techniques to become standard practice.
References


[IEEE] All referenced IEEE material is used with permission.


References (Cont.)


References (Cont.)

Additional Reading:


