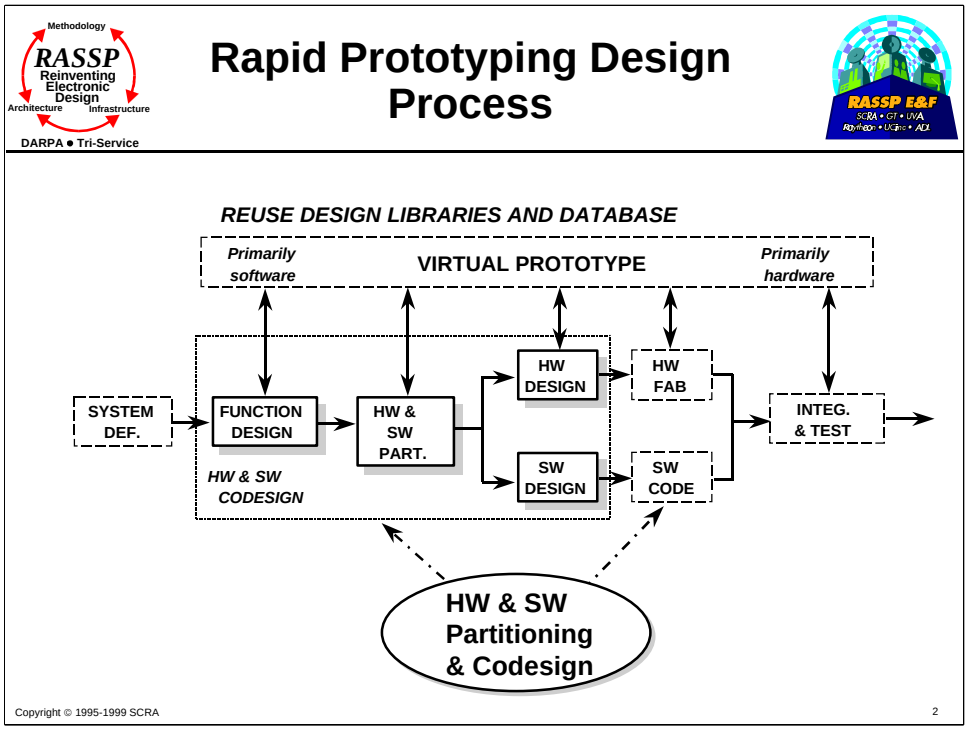
DARPA ● Tri-Service

Hardware/Software Codesign Overview

RASSP Education & Facilitation Program
Module 14

Version 3.00



This slide shows where the Hardware/Software Codesign and Partitioning process fits into the RASSP design flow.

RASSP Reinventing Electronic Design — Methodology, Architecture, Infrastructure
DARPA • Tri-Service

Module Goals

RASSP E&F — SCRA • GT • UVA • Raytheon • UCSB • ADI

- **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**

The goals of this module are to present what hardware/software codesign and partitioning is, what the benefits of truly integrated codesign are, and how industry and research groups are attempting to automate parts of the codesign process.

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**

DARPA ● Tri-Service

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**

 6

The common definitions for HW/SW codesign is presented above. The two key concepts involved in codesign are concurrent development of HW and SW, and integrated design. Integrated design allows interaction between the design of HW and SW. Codesign techniques using these two key concepts take advantage of design flexibility to create systems that can meet stringent performance requirements with a shorter design cycle.

[DeMicheli97],[Franke91],[Kumar95],[Subrahmanyam93]

RASSP Methodology Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

Motivations for Codesign

RASSP E&F SCRA • GT • UVA Raytheon • UCTec • ADI

- **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**

 7

The major factor driving the need for hardware/software codesign is the fact that most systems today include both dedicated hardware units and software units executing on microcontrollers or general purpose processors.

The increasing use of programmable processors being used in systems that formerly may have been all hardware, the availability of cheap microcontrollers for use in embedded systems, the availability of processors cores that can be easily embedded into an ASIC design, and the increased efficiency of higher level language (C and C++) compilers that make writing efficient code of embedded processors much easier and less time consuming are all factors that are increasing the amount of software in embedded systems.

[DeMicheli97]

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**

This module will concentrate on the use of codesign in the development of embedded systems. A number of technologies have advanced recently enabling codesign to become feasible:

(1) High-level hardware synthesis capabilities of improved design automation tools.

(2) ASIC development allows complex algorithms to be implemented in silicon quickly and inexpensively .

(3) RISC technology has allowed traditional hardware functionality to be implemented in software

[Kumar95].

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**

 9

The “best” solution for performing hardware/software partitioning and codesign depends on the type of system being designed. Therefore, it is necessary to talk about ways to categorize hardware/software systems. Above are listed three major ways in which these systems can be categorized.

[DeMicheli97]

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 of concurrent with execution (called *evolvable* systems)**

The categories of the codesign problem can best be aligned with the application domain of the system being designed. Here are listed some distinguishing characteristics of each system that influence the codesign problem.

[DeMicheli97]

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

The codesign problem consists of specifying the system (typically in a behavioral form), in a representation that is suitable for describing either hardware or software, partitioning the system into either hardware or software, scheduling the execution of the system's tasks to meet any timing constraints, and modeling the system throughout the design process to validate that it meets the original goals and functionality.

[DeMicheli97]

Embedded Systems

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, as defined above, have been the impetus for much of the interest in hardware/software codesign.

Form factor measures include size, weight, and power consumption.

[Subrahmanyam93], [Wolf94]

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

These are trends in embedded systems that have brought about the need for codesign techniques to meet design constraints.

[Wolf94]

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

Early uses were in banking and transaction processing applications.

In early embedded systems, expensive hardware justified the relatively high cost of designing and maintaining system software.

Later microprocessors made low-cost embedded systems a realistic possibility.

[Wolf94]

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

Because of the increasing capacity of digital system implementation technologies, the complexity of embedded systems is growing at an extremely fast rate. This growth is impacting the scope of the codesign problem.

Techniques to Support Complexity Management

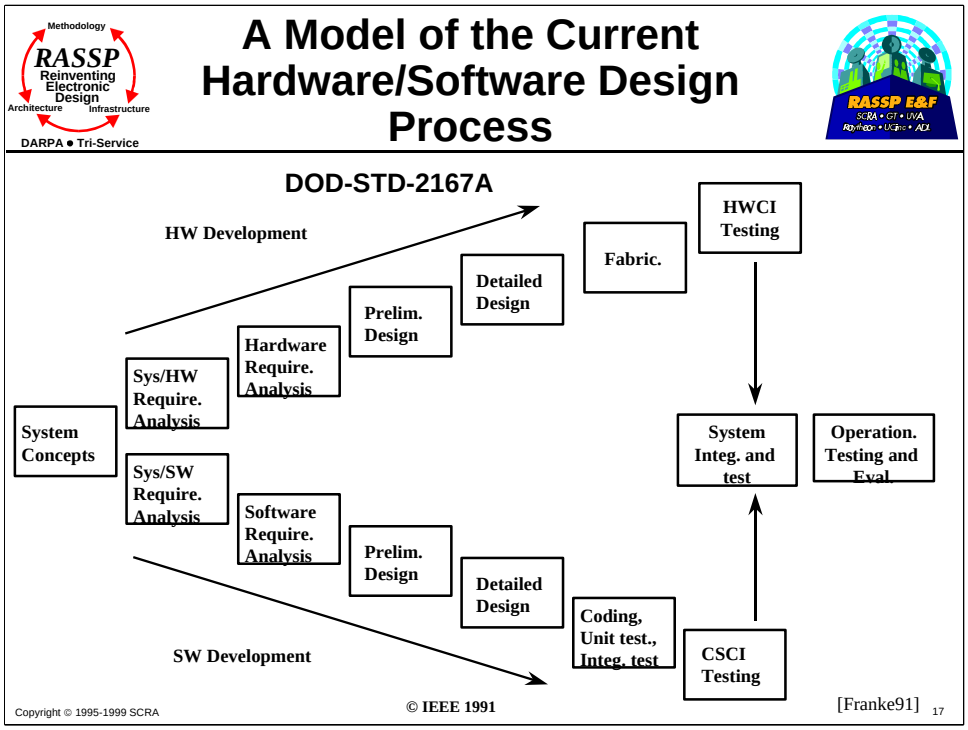
- **Delayed HW/SW partitioning**
  - **Postpone as many decisions as possible that place constraints on the design**
- **Abstractions and decomposition techniques**
- **Incremental development**
  - **"Growing" software**
  - **Requiring top-down design**
- **Description languages**
- **Simulation**
- **Standards**
- **Design methodology management framework**

Several ways to help manage the complexity of a design are listed above.

Growing software:

Always having a version of the code that runs even at the beginning of the design process when the code does very little.

[Thimbleby88]



This is a model of the current codesign process from DOD standard 2167. Note that in this model, after initial partitioning, HW and SW remain separate entities, with no further communication until integration. This causes serious problems with the implementation of the hardware/software interfaces and integration to go undetected until the final integration process - which is often too late to make changes in the architecture necessary to fix them.

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**

 18

The separate development of HW and SW restricts the ability to study HW/SW tradeoffs. A "Hardware First" approach is often pursued with the following characteristics:

(1) Hardware is specified without understanding the computational requirements of the software.

(2) Software development does not influence hardware development and does not follow changes made to hardware during its design process.

With this type of process, problems encountered as a result of late integration can result in costly modifications and schedule slippage.

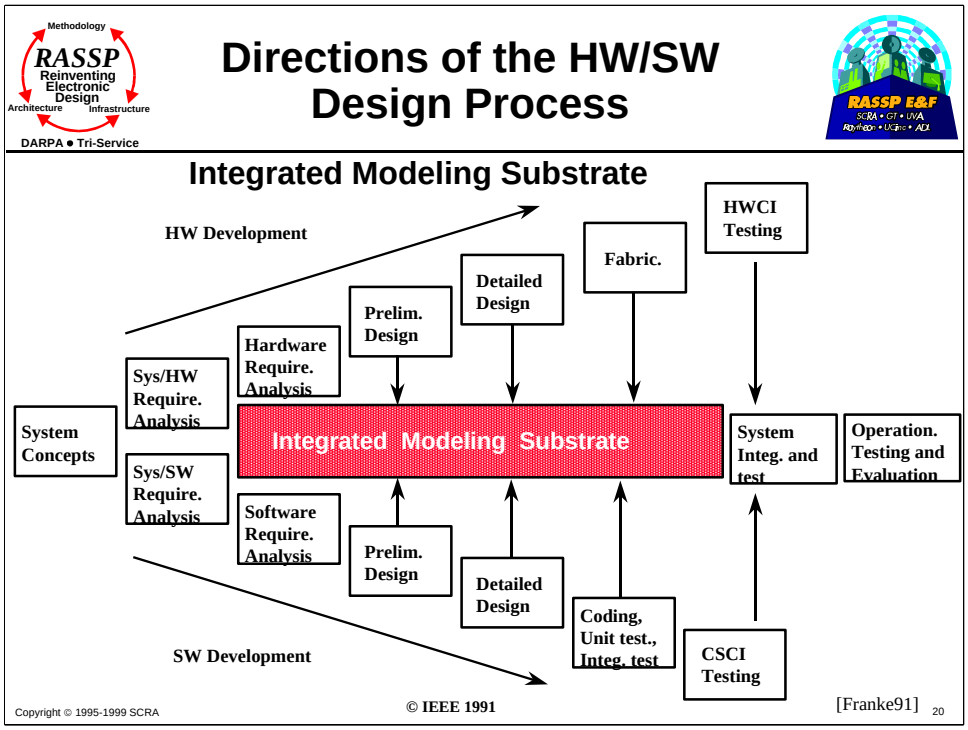
[Franke91], [Thimbleby88], [Turn78]

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

Some common misconceptions about hardware and software design which adversely impact the current, disconnected hardware/software design process.

[Turn78]



This figure shows one of the requirements for an efficient codesign process - an integrated substrate for modeling both the hardware and software and their interactions. The integrated modeling substrate allows for incremental review throughout the design process, with interaction between hardware and software.

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)**

Partitioning - the process of determining which functions to implement in hardware and which in software.

Iterative Partitioning Techniques: Repeated HW/SW partitioning is used to improve the system performance.

By switching certain functions to hardware and some to software, the speed, cost, and other performance metrics can be affected. Partitioning is done over and over, moving the partition until optimal performance is obtained.

Continuous/incremental evaluation: Evaluating the hardware and software designs at several stages of the design process allows interaction between the HW and SW designs at the earliest stages.

Continuous/incremental evaluation is facilitated in the integrated modeling substrate by allowing HW and SW changes to be taken into consideration in both design paths at early stages rather than waiting until integration. This makes the integration process much smoother.

[Franke91], [Kumar95], [Thimbleby88], [Turn78]

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**

The benefits of an integrated modeling substrate have been stated before. The system must also include a validation methodology to insure that the system meets its initial requirements. Some codesign environments are attempting to meet this validation requirement using formal verification techniques on the initial unified representation of the system. Others use the integrated modeling substrate to provide simulation-based validation.

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**

Advances in the automation of the design of hardware have influenced the development of tools to automate the design of software, and visa versa. This represents an opportunity to exploit common techniques in the codesign process. Some research efforts that exploit this cross-fertilization are discussed later in the module.

[Smith86]

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

DARPA ● Tri-Service

VLSI DESIGN → **SOFTWARE ENGINEERING**

- ***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**

 24

Several components of EDA (electronic design automation) tools for VLSI design have been transferred into tools for automating software design:

- Designer support rather than automated synthesis of design
- Graphics-driven designs
- A central repository of design information
- Tools for early assessment of correctness and quality of the design.

The EPOS system is an example of a system for both HW and SW design that uses a hierarchical graphics-driven design.

The CADES system emphasizes management and control of design environment using a central database.

[Smith86]

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

DARPA ● Tri-Service

SOFTWARE ENGINEERING → **VLSI DESIGN**

- ***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**

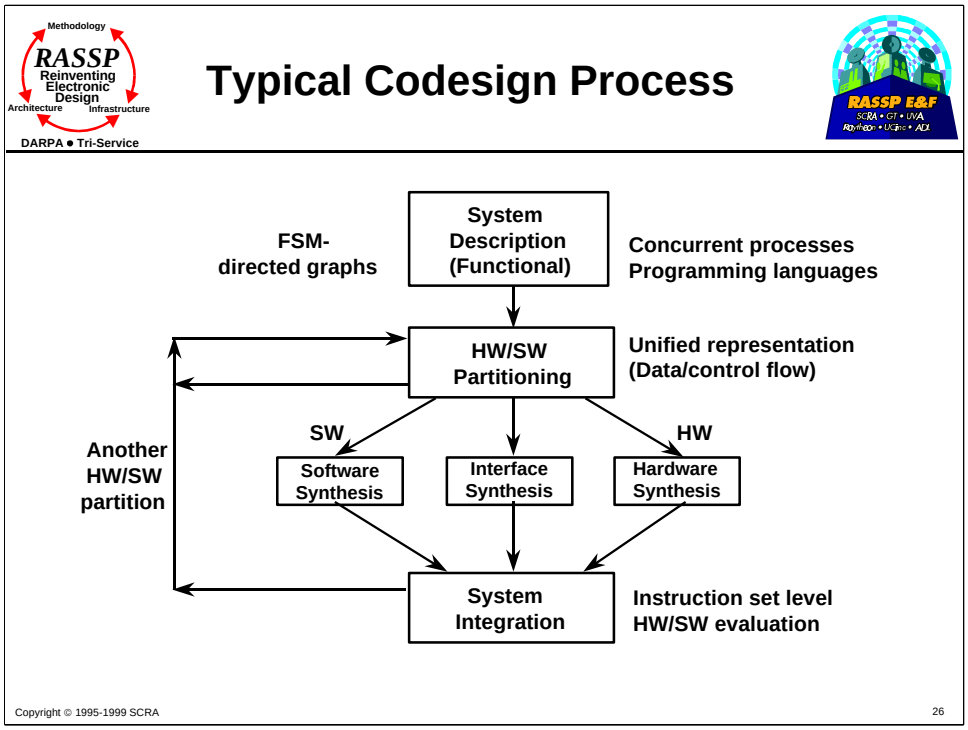
25

Compiler-like transformations and techniques have been applied to high-level hardware synthesis capabilities, such as dead code elimination and loop unrolling.

Some efforts have tried to address design change management issues. Information hiding and program families have been used to try to minimize the impact of change in the VLSI design process.

The concept of design families involves characterizing a set of programs as a family “whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members.” Family similarities during all stages of design will lead to designs that are relatively easy to modify.

[Kumar95], [Smith86]



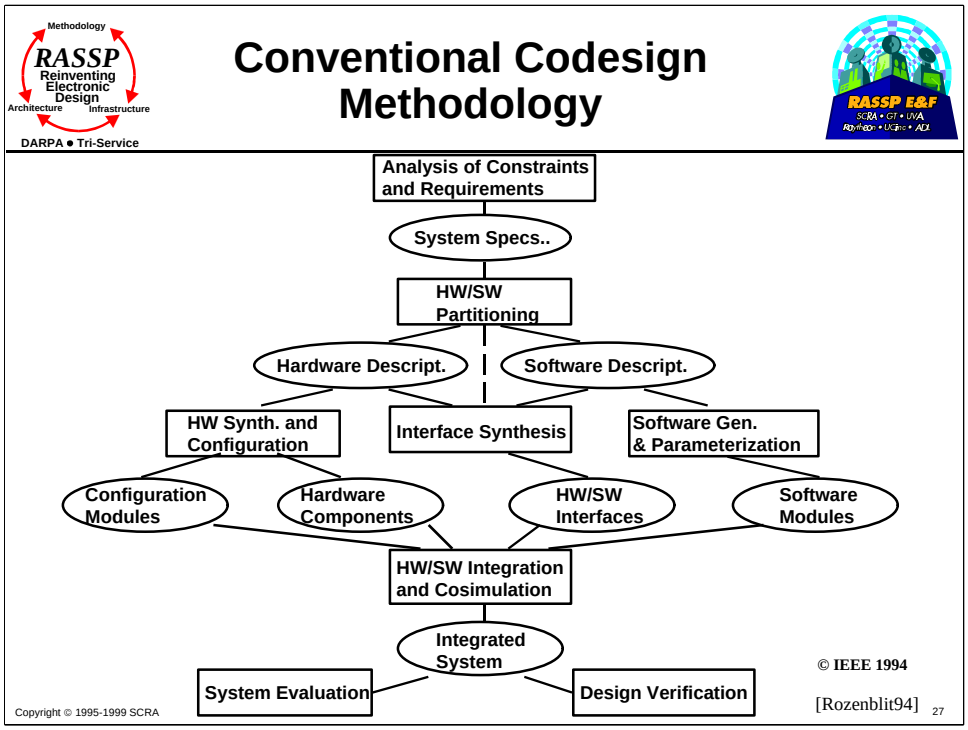
The codesign process starts with an architecture independent description of the system functionality. This description is independent of HW and SW, and several system representations may be utilized, e.g. finite state machines (FSMs).

The system is then described by means of a programming language, which is next compiled into an internal representation such as a data control flow description. This description serves as a unified system representation that can represent HW or SW.

HW/SW functional partitioning is performed on this unified representation. After this step has been completed, HW, SW and related interfaces are synthesized.

Evaluation is then performed. The partitioning process is iterative, and If the evaluation does not meet required objectives, another HW/SW partition is generated and evaluated.

[Kumar95]



This is a general view of codesign. It is not taken from a specific approach used by one group. Rather, it reflects a combination of several approaches presented recently in literature.

Note partitioning stage and the integration phases common to all codesign methodologies. Codesign is still a relatively new, changing approach, so there is not one set standard for how it must be done. Many variations exist.

[Rozenblit94, p.4]

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**

One of the major features of a good codesign process is that it allows faster exploration of the design space. This includes analyzing different configurations for the overall system (irrespective of its implementation) AND analyzing different hardware/software partitions for a given system configuration.

[Subrahmanyam93].

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**

Lack of a standardized representation: The industry lacks a common model or standard for unified exchangeable design representations which would greatly enhance usage of codesign.

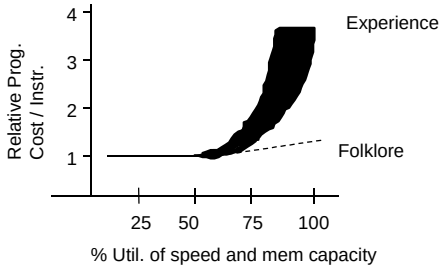
Lack of good validation and evaluation methods: Very few comprehensive tools are available for hardware/software cross-specification, development, simulation, integration, and test. However, efforts are underway to provide them.

[Buchenrieder93].

Issues and Problems: Integration

- **Errors in hardware and software design become much more costly as more <u>commitments</u> are made**
- **"Hardware first" approach often compounds software cost because software must compensate for hardware inadequacies**

Software Cost Impact of Inadequate Hardware Resources



In software, problems found in development testing are at least one order of magnitude more costly to fix than those found during requirements specifications. Therefore, it is important that the system be able to be validated as the design progresses. The most common way to perform this validation is through simulation.

[Boehm73], [Terry90], [Thimbleby88], [Turn78]

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**

One of the keys to a GOOD hardware/software codesign process is a unified representation the allows the functionality of the system (at various levels of abstraction) to be specified in a manner that is “unbiased” towards either a hardware or software implementation.

Again, this description must be validated to ensure that it meets the original system specifications. This validation typically happens through simulation although at a high level, formal techniques can sometimes be applied.

[Kumar95]

Current Abstraction Mechanisms in Hardware Systems

DARPA ● Tri-Service

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**

This slide discusses the process of abstraction - describing the system in only as much detail as is absolutely necessary to perform the analysis of current interest.

[McFarland90]

Abstractions in Modeling: Hardware Systems

Level	Behavior	Structure	Physical
PMS (System)	Communicating Processes	Processors Memories Switches (PMS)	Cabinets, Cables
Instruction Set (Algorithm)	Input-Output	Memory, Ports Processors	Board Floorplan
Register-Transfer	Register Transfers	ALUs, Regs, Muxes, Bus	ICs Macro Cells
Logic	Logic Equns.	Gates, Flip-flops	Std. cell layout
Circuit	Network Equns.	Trans., Connections	Transistor layout

Start here

Work to here

[McFarland90]

34

This chart illustrates various levels of abstraction possible for hardware systems along three description domains, i.e. behavior, structure, physical.

Current Abstraction Mechanisms for Software Systems

DARPA ● Tri-Service

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**

Ex: High-level language such as C is used by a developer. Programs are written for a "virtual machine" that understands the language's instruction set.

[Kumar95], [Tanenbaum87]

Abstractions for Software Systems

DARPA ● Tri-Service

Virtual Machine Hierarchy

- Application Programs
- Utility Programs
- Operating System
- Monitor
- Machine Language
- Microcode
- Logic Devices

Virtual machine hierarchy:

Multiple layers of software on top of hardware are shown for a typical computer system.

Operating system represents one level of a virtual machine.

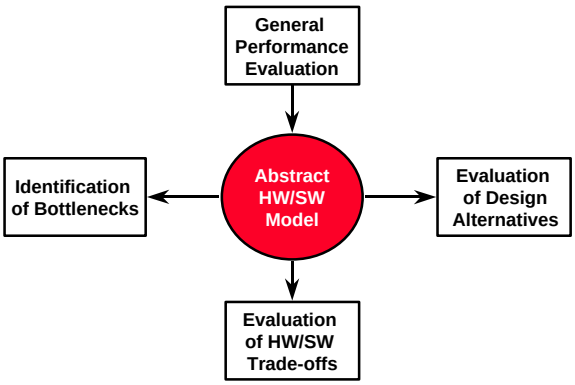
Higher levels such as utility programs (compilers, editors, interpreters, etc) and application programs can also be viewed as virtual machines.

[Tanenbaum87]

Abstract Hardware-Software Model

DARPA ● Tri-Service

Uses a unified representation of system to allow early performance analysis



An abstract HW/SW model is developed to promote early performance analysis. Using unified representation based on data/control flow concepts, the abstract HW/SW model supports general performance evaluation, the identification of bottlenecks, the evaluation of HW/SW tradeoffs, and the evaluation of design alternatives. The model can be used to assess the consequences of various HW/SW partitioning decisions before committing to a particular design.

[Kumar95]

Examples of Unified HW/SW Representations

Systems can be modeled at a high level as:

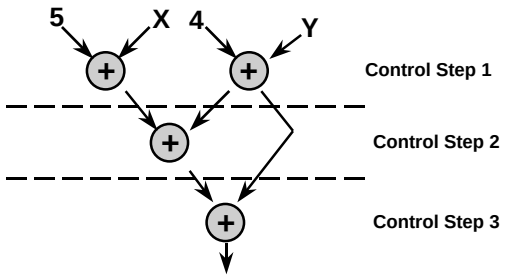
- **Data/control flow diagrams**
- **Concurrent processes**
- **Finite state machines**
- **Object-oriented representations**
- **Petri Nets**

There are numerous methods that are candidates to be used for a unified representation. Most all of them have been tried in one codesign system or another with varying levels of success. Typically the methods are more suited to systems of a certain type, e.g., data flow diagrams are more suited to data driven applications like Digital Signal Processing (DSP) systems.

Unified Representations (Cont.)

- **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:



This figure presents data flow graphs in more detail. DFGs are used in many high level specification tools such as Ptolemy, SES Workbench, etc.

[Kumar95]

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**

This slide discusses two other mechanisms that have been used for unified representations, concurrent processes and finite state machines. Both of these representations are more suited to control dominated applications such as a real-time, reactive control system.

[Chiodo92]

RASSP Reinventing Electronic Design — Methodology, Architecture, Infrastructure

DARPA ● Tri-Service

Unified Representations (Cont.)

RASSP E&F — SCRA • GT • UVA • Rockwell • Logic • ADI

- **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**

Object oriented representations can be used as a unified representation. They do, however suffer somewhat from complexity and are more ideally suited to describing software than hardware.

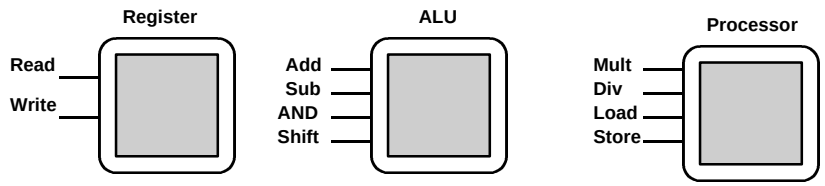
[Kumar95]

Unified Representations (Cont.)

Object-oriented representation

Example:

3 Levels of abstraction:



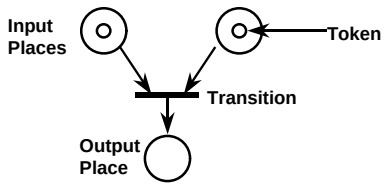
This is a simple example of three different levels of hardware abstraction that can be described in an OO representation.

[Kumar95]

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:



Petri Nets have been used to describe both hardware and software and therefore are a candidate for a unified representation. They also suffer somewhat from a complexity issue (many places and transitions are necessary to model a fairly simple system). Petri Nets were not developed to describe the complete functionality of a system, and thus are not very applicable for low-level functional descriptions.

[Kumar95]

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**

45

Partitioning the system into dedicated hardware components and software components executing on Instruction Set Processors is a vital part of the codesign process.

Partitioning requires the use of performance and other metrics to assist the partitioner (either human or automated) in choosing from among several alternative hardware and software solutions.

Because there are multiple metrics that must be optimized at the same time, finding an *optimum* partition is an NP-hard problem.

[Gajski94].

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**

 46

Issues of hardware implementation vs software implementation must be addressed when performing partitioning. There are pros and cons to both hardware and software implementations. The system requirements and performance goals must be examined to determine which criteria are most critical for the particular system.

In general, HW implementation supports parallel execution of operations while incurring the cost of hardware fabrication. Software implementation is generally slower than custom hardware implementation, but does not require high cost of fabrication. Similarly, partitioning may be driven by schedule requirements in which there is not time to build custom hardware.

[DeMicheli93].

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)***

 47

There are two basic approaches that most designers use when performing partitioning.

They either start with all operations in software and move some into hardware (when speed is critical) or they start with all operations in hardware and move some into software. Different design environments support one or the other. For example Cosyma, a cosynthesis approach to design, starts with all functions generated in software and then moves operations to hardware only as time constraints are violated.

A team at the University of Braunschweig, Germany, explored codesign tradeoffs in systems that were originally implemented in software (written in C)

- A partitioning program identified the computational bottlenecks and migrated the corresponding functions to application-specific hardware
- With system-level partitioning, a critical loop which took up 90% of the software execution time for a HDTV Chromakey algorithm was implemented in hardware (as a 17,000 gate ASIC) leading to a 3X speedup

A team at Stanford University explored migrating hardware components to software routines

- Identifying non-critical tasks which can be migrated from hardware to software implementations lead to significant size and cost reductions without reducing performance
- A system model which specified performance requirements in terms of latency and data-rate constraints was used to support the partitioning
- A 20% reduction in gate count was achieved

[DeMicheli94]

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

Functional partitioning allows the system to be partitioned into hardware and software components.

It is analogous to Structural Partitioning in which the structure of a system is refined into lower level hardware components. However, in Structural Partitioning, functionality cannot be moved from hardware to software.

[Gajski94].

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**

Metrics must be used to guide the partitioning process. The type of metrics used depends a great deal on the level of description of the system.

[DeMicheli93].

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**

Early binding is widely used in industry because it supports complete planning of the design cycle. With this method, hardware/software partitioning decisions have to be made very early in the design.

Late binding because of its flexibility, provides greater opportunity for performing hardware/ software tradeoffs. Therefore, late binding generally results in a more optimal partition.

[Buchenrieder93].

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**

HW/SW partitioning algorithms are usually targeted to systems in which only a few operations need specialized hardware.

[Wolf94].

HW/SW Partition Formal Definition

- **A hardware/software partition is defined using two sets *H* and *S*, where $H \subset O$, $S \subset O$, $H \cup S = O$, $H \cap S = \phi$**
- **Associated metrics:**
  - **Hsize(*H*) is the size of the hardware needed to implement the functions in *H* (e.g., number of transistors)**
  - **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, ... C_m), where C_j = {*G*, *timecon*}, indicates the maximum execution time allowed for all the functions in group *G* and $G \subset O$**

This slide presents the formal definition of a hardware/software partition.

[Vahid94].

Performance Satisfying Partition

- A performance satisfying partition is one for which performance($C_j.G$) ≤ $C_j.timecon$, 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))

Therefore, the problem is to map functions to either hardware or software in such a way that we find the minimal hardware for which all performance constraints can be met.

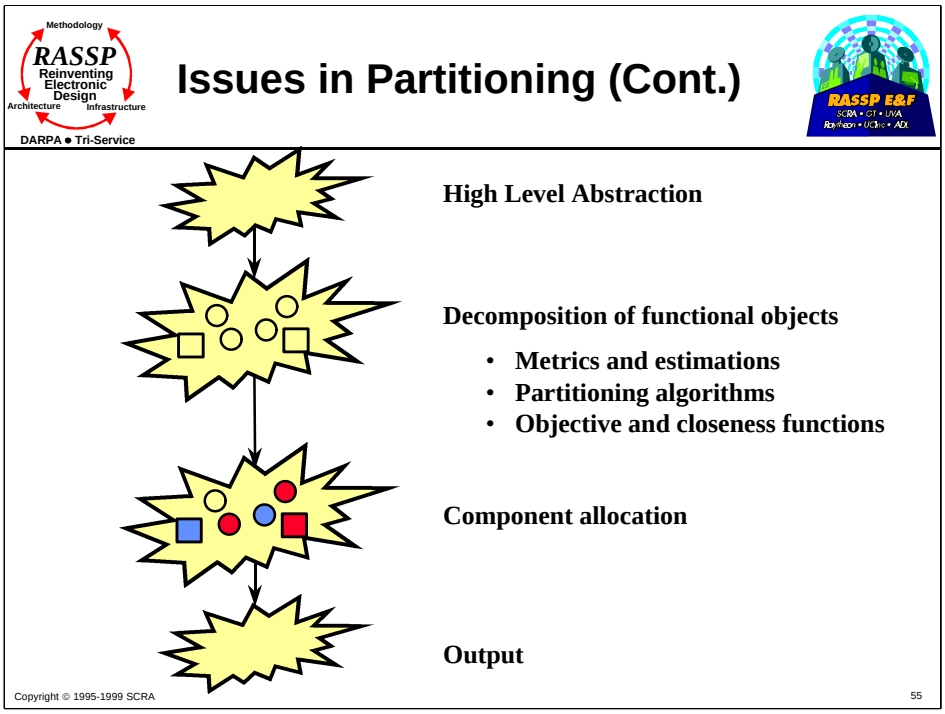
[Vahid94].

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

There are a number of issues that influence the partitioning problem, both in its difficulty, and in the quality of solutions.

[Gajski94].



The material is presented in the sequence in which we encounter these issues in a typical partitioning process:

First, the specification abstraction level defines the input in terms of the abstraction level of the conceptual model.

We are then concerned with the granularity of the functional objects into which the input is decomposed.

Metrics, partitioning algorithms, objective and closeness functions are used to determine which partition to implement.

The system component allocation process chooses components, from among those available, to implement the partition.

Finally, we have the output, which is a fully implemented system.

[Gajski94].

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**

 56

Design usually begins with higher abstraction levels, as designers initially conceptualize at those levels.

Thus, different levels of input to partitioning techniques represent different amounts of design already done before partitioning is performed.

[Gajski94].

Specification Abstraction Levels (Cont.)

- **Register transfers**
  - **The transfers between registers for each machine state are described**
- **Structure**
  - **A structural interconnection of physical components**
  - **Often called a netlist**

Design then progresses to lower levels such as these.

[Gajski94].

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**

Note that it is the system functionality that is being partitioned here in order to achieve a better allocation and assignment to hardware or software. That is, a number of objects in a partition defined here may be assigned to the same hardware or software later.

[Gajski94].

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.

An integral part of the partitioning problem is allocating portions of the system to actual components for their implementation. Obviously, the system “tasks” must be allocated to components that are capable of performing them.

[Gajski94].

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**

Metrics must be defined to determine a partition's relative cost vs. other potential partitionings. Obviously, some metrics, such as execution time of a given task on a specific processors, are impossible to measure precisely until a final implementation is made. Therefore, accurate, fast "cost" estimation is mandatory for a good partitioning algorithm.

[Gajski94].

Metrics in HW/SW 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**

A consideration of metrics is only important once the requirements are satisfied.

The hardware size metric is defined as improved when there is a *reduction* in amount of HW.

The performance metric is defined as improved when there is an *increase* in the amount of HW and a corresponding *decrease* in the amount of SW.

[Gajski94].

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***

There are two basic approaches to computing metrics. The first is to create a detailed implementation and directly measure the metrics of interest. The second is to *estimate* the given metric from the abstract system model in use at the time.

[Gajski94].

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., Objfct = k1 * area + k2 * delay + k3 * power**
  - **Because constraints always exist on each design, they must be taken into account**
    - **e.g Objfct = k1 * F(area, area_constr)**
      + k2 * F(delay, delay_constr)
      + k3 * F(power, power_constr)

Varying *weights* for area, timing, and power constraints may be used to reflect their relative importance in each different system being designed.

[Gajski94].

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

For most partitioning algorithms based on multiple metrics, finding an *optimal* partition is an NP-hard problem. Therefore, heuristics must be employed to reduce the complexity and find a “good enough” partition.

The other option, of course, is to leave the partitioning up to the user. This is the approach used in several noteworthy HW/SW codesign research efforts.

[Gajski94].

Partitioning Algorithm Classes

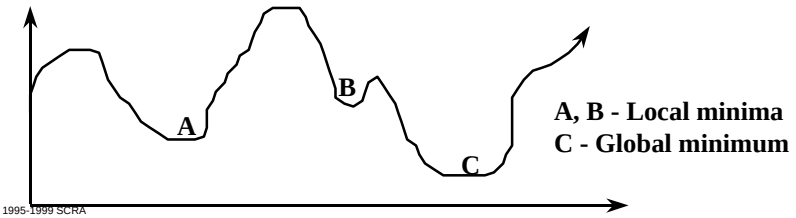
- **Constructive algorithms**
  - **Group objects into a complete partition**
  - **Use closeness metrics to group objects, hoping for a good partition**
  - **Spend computation time constructing a small number of partitions**
- **Iterative algorithms**
  - **Modify a complete partition in the hope that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

A partitioning algorithm can be classified into several general categories.

[Gajski94].

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



In the graph above, C represents a solution which would consume fewer resources (in HW and SW) than A or B, yet it might be very difficult to find, and solution A may be “good enough.”

Multivariate optimization is a much-studied problem in CS. The interested reader should investigate this area to understand the problems inherent in optimal partitioning.

[Gajski94].

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**

Greedy algorithms only go “down hill” and thus are likely to get stuck in local minima.

An important advantage of hill-climbing algorithms over greedy algorithms lies in the hill-climbing algorithms’ ability to escape local minima. With this ability, they are more likely to find the global minimum and provide a better solution. They are however, more computationally complex.

[Gajski94].

Output Issues in Partitioning

- **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**

Potential roles for the result of HW/SW partitioning include:

1) Subsequent use as a functional specification for humans who must implement each component.

2) Subsequent use as the input to a synthesis tool.

[Gajski94].

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.)**

The decision-making sequence chosen will impact the final partition obtained.

It is advantageous to use the sequence that yields good results for a particular design goal, say, maximizing performance.

[Gajski94].

Comparing Partitions Using Cost Functions

- A *cost function* is a function 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

  Part_Alg(*H*, *S*, *Cons*, *I*, Cost( ))

  which returns a partition *H'*, *S'* such that

  Cost(*H'*, *S'*, *Cons*, *I*) ≤ Cost(*H*, *S*, *Cons*, *I*)

The formal definition of cost functions and iterative partitioning algorithms. If Cost(*H'*, *S'*, *Cons*, *I*) ≤ Cost(*H*, *S*, *Cons*, *I*) is true, a better partition has been found.

[Vahid94].

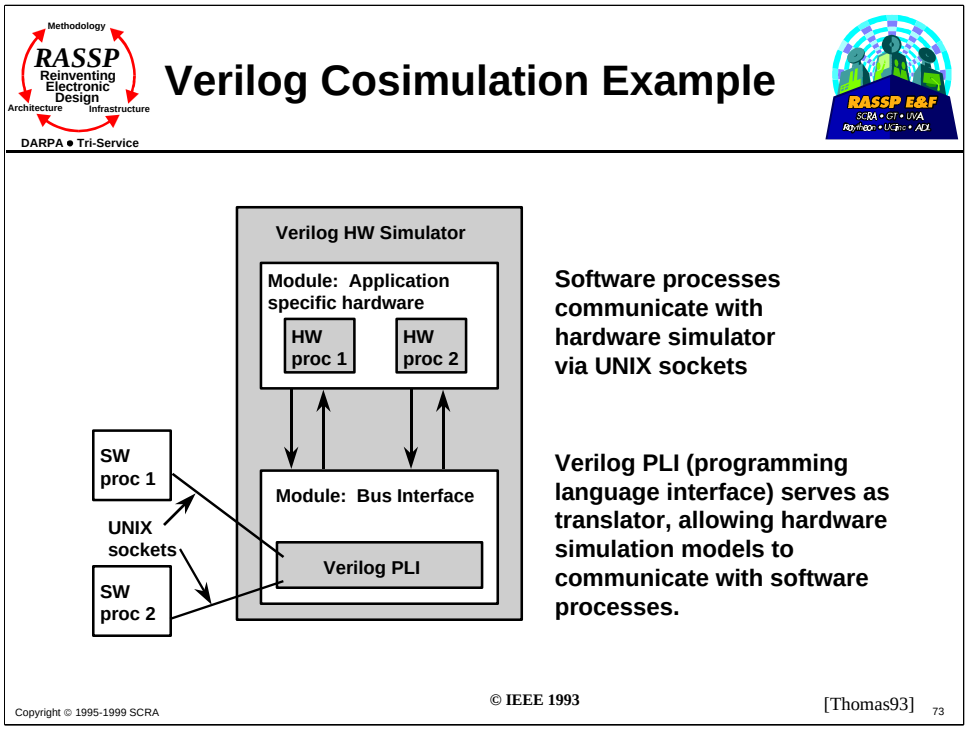
Module Outline

- 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)

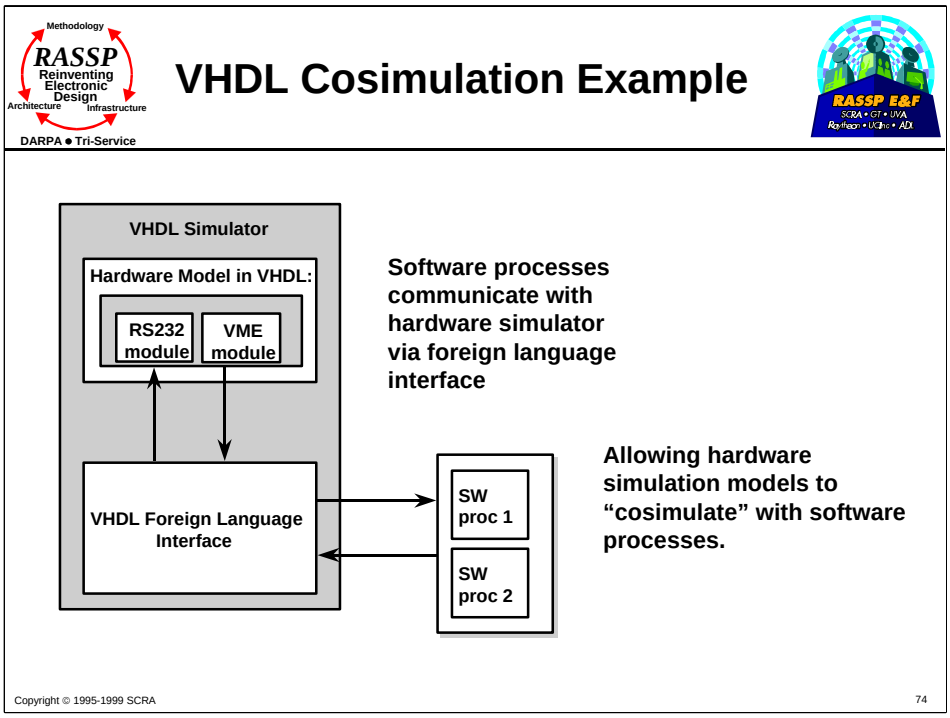
Cosimulation in the codesign context, is the simulation of a system's hardware and software in the same environment. Obviously, when the entire system is specified in the unified representation, a single simulation environment can be used. However, often after partitioning, the hardware and software are represented using different languages and modeling paradigms, thus cosimulation is necessary to validate the system through simulation.



A team at Carnegie-Mellon University has developed a cosimulation environment based on the use of the Verilog hardware description language to describe the hardware and C or C++ to describe the software.

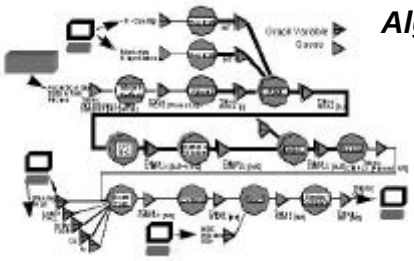
The Verilog simulator is used to perform behavioral simulation of the system hardware processes. The software processes run as separate Unix processes and communicate with the hardware simulator by means of the BSD (Berkeley Software Distribution) Unix socket facility. Many aspects of the system are hidden by the abstraction used for HW/SW interaction.

In the Verilog simulation environment, one or more modules comprise the application specific portion of the hardware. A separate module acts as the bus interface. The bus interface module translates the socket activity into the appropriate simulation events. The routines that do the translation are implemented in C and linked to the Verilog simulation environment through the Verilog PLI.



A similar mechanism can be used to develop a cosimulation environment using VHDL and C++.

VHDL-C Based HW/SW Cosimulation for DSP Multicomputer Application

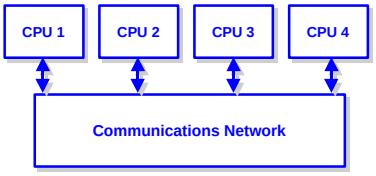


Algorithm - C

Scheduler - C

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

Architecture - VHDL



VHDL-C Based HW/SW Cosimulation for DSP Multicomputer Application

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

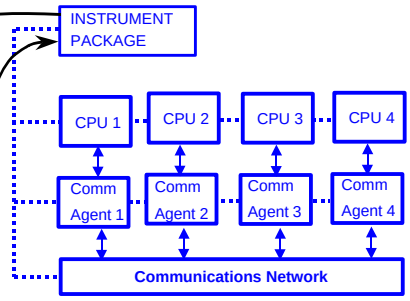
Algorithm/ Scheduler

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

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

VHDL Simulator

Architecture Model



Model Continuity Problem

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**

Another problem with the current HW/SW design process involves the model continuity problem. High-level system models have not been useable as the design progresses to lower, more detailed design. This lack of continuity prevents the designer from using the analysis performed with the high-level model at the more detailed stages of design. This lack of model continuity is seen in both hardware and software systems.

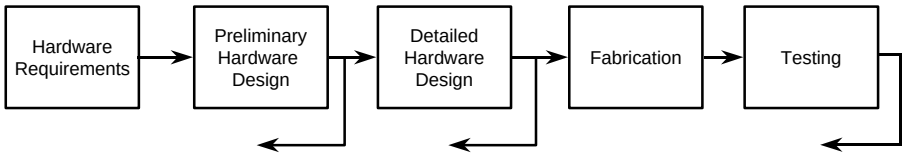
[Franke91]

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



The traditional hardware design process is a serial, waterfall process with few feedback loops to iterate over the design space.

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**

High-level hardware synthesis tools are often used with high-level preliminary designs to develop hardware structures from:

(1) Behavioral descriptions

(2) System design constraints.

Lower-level synthesis tools can then be used to take hardware structure and derive algorithmic, register-transfer, logic, and circuit-level designs.

[Kumar95]

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 is the automated mapping of a behavioral description onto a specific hardware implementation.

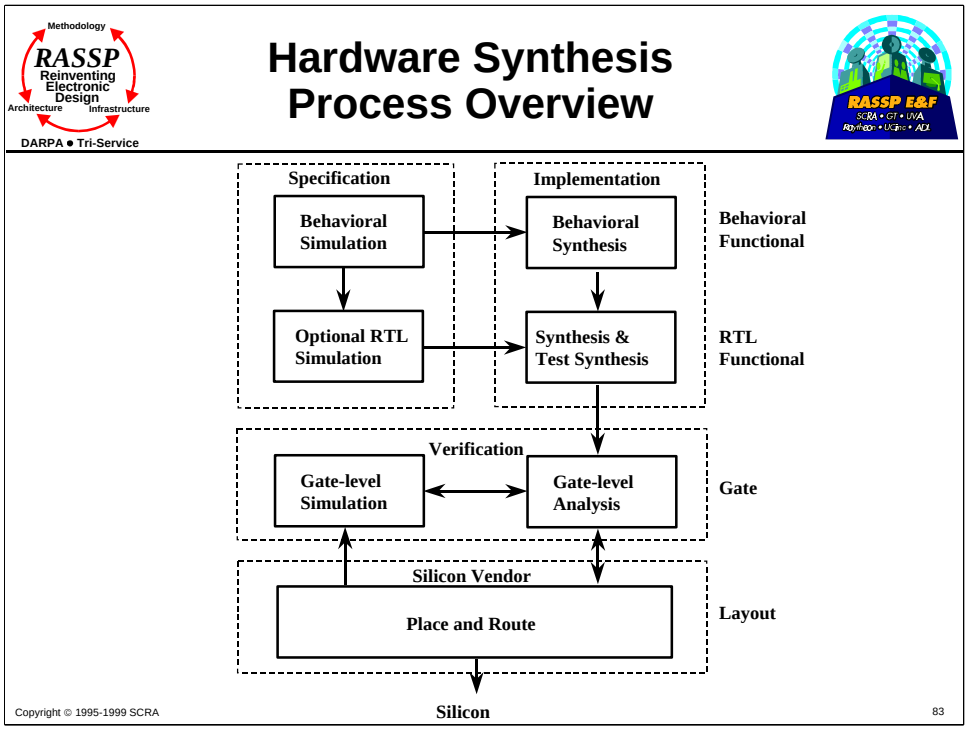
[Parker84]

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)**

These terms and definitions describe the levels of hardware to which hardware synthesis tools operate.

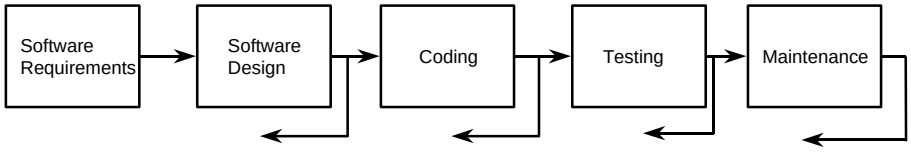
[Parker84].



This is a generic overview of the hardware synthesis process.

Software Design Methodology

Software Design Process: Waterfall Model



The traditional software design process can also be described by a serial, waterfall process.

[Jalote91], [Kumar95]

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**

Requirements Phase - consists of both defining the spec. for the system and defining the requirements for system analysis.

- Requirements phase produces: Software Requirements Specification (SRS) document.
- Must be complete and clear because changes in spec. can be very costly once design has progressed to later stages.

Coding is in high-level language: C, C++ for example.

Multiple testing levels consist of:

- Unit testing - individual module alone
- Integration testing - several interconnected modules
- System testing - entire software subsystem in intended hardware environment
- Regression testing - if changes made to model
- Acceptance testing - performed by user.

[Jalote91]

RASSP Reinventing Electronic Design
Methodology
Architecture
Infrastructure
DARPA ● Tri-Service

Software Synthesis

RASSP E&F
SCRA • GT • UVA
Raytheon • Logic • ADI

- **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**

Software synthesis is the automated mapping of functionality into executable code.

[Kumar95]

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

The use of software synthesis is become more prevalent for the same major reasons hardware synthesis is, design (coding) time is less, reusability is higher and some more “correct by construction” techniques can be applied.

Software Synthesis Categories

- **Language compilers**
  - **ADA and C compilers**
  - **YACC - yet another compiler compiler**
  - **Visual Basic**
- **Domain-specific synthesis**
  - **Application generators from software libraries**

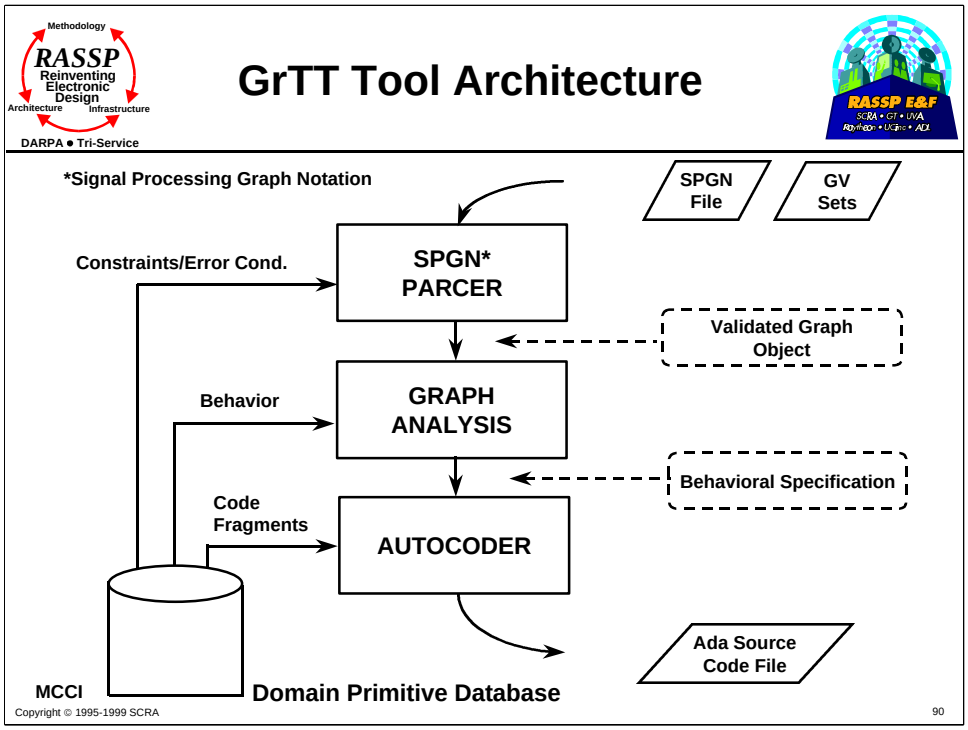
Software synthesis can mean complex language compilers, or tools that generate actual software code from abstract descriptions such as DFGs. However, the applicability of these latter tools seems to be very application specific.

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**

Here are some examples of software synthesis systems, mostly of the latter type discussed on the previous page.

[Terry90]



This figure presents the architecture of the GrTT tool. It accepts the SPGN (Signal Processing Graph Notation - a textual representation of the PGM data flow graph) representation of an application graph and generates source code that can then be compiled to the specific target architecture.

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**

The automated synthesis of the interfaces between system components implemented in hardware and software is a big issue. Several of the codesign research systems described later in this module include techniques for automating the creation of these interfaces.

Interface Synthesis Approaches

- **Typical approaches use standard interface schemes**
  - **memory-mapped**
  - **serial port**
  - **parallel port**
  - **self-timed**
  - **synchronous**
  - **blocking**

There are many techniques for interfacing between software executing on a processor and dedicated hardware outside that processor, and thus, there are many different ways to synthesize hardware/software interfaces.

RASSP Reinventing Electronic Design
Methodology
Architecture
Infrastructure
DARPA ● Tri-Service

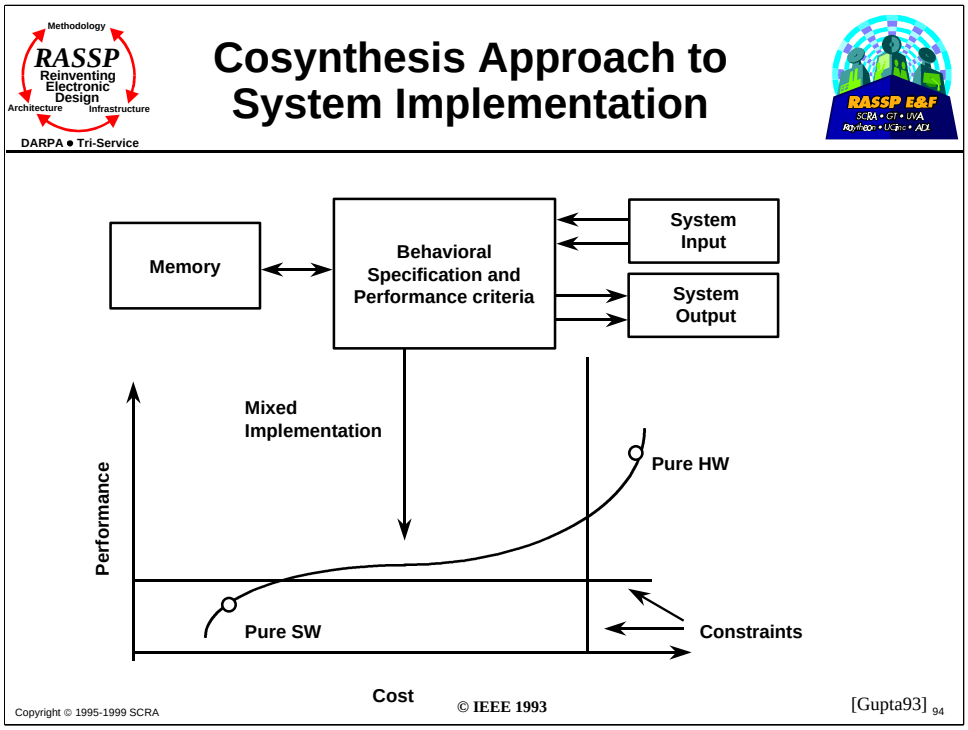
Cosynthesis

RASSP E&F
SCRA • GT • UVA
Raytheon • Logic • ADI

- **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" has come to mean the concurrent synthesis of the hardware and software portions of the system, trading-off various implementation techniques between them to arrive at a more "optimum" overall solution.

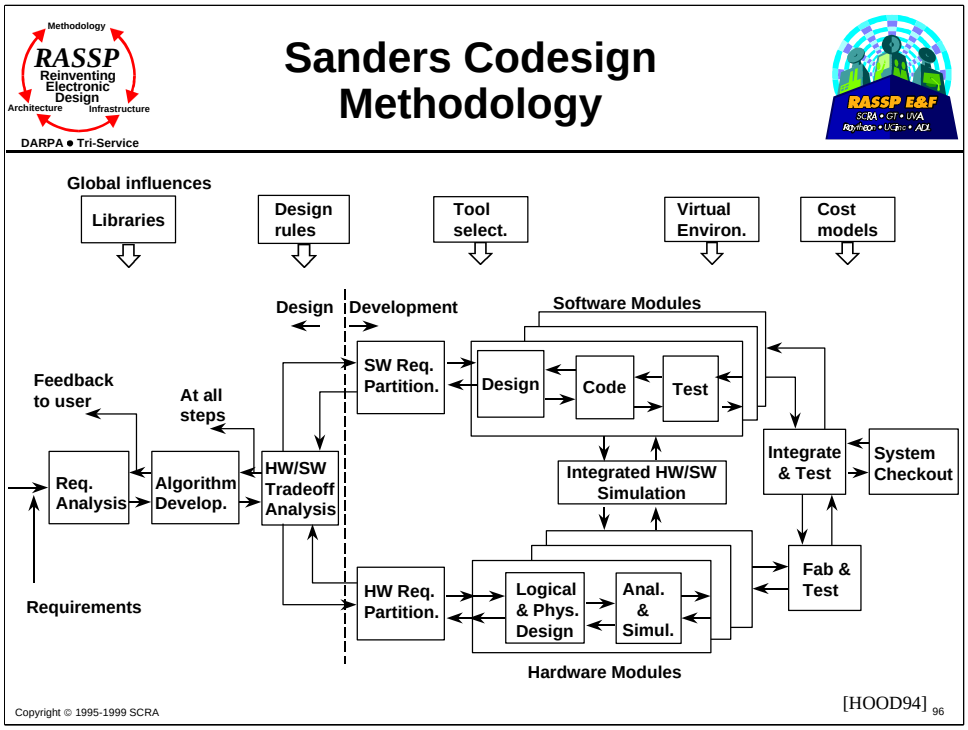
[Gupta92]



Above figure represents design trade-offs common in synthesis while also creating a cost-effective system. While a pure HW system may fulfill all performance needs, it may not meet constraints such as cost. The pure SW implementation may meet the cost constraint, but not the performance goals. The cosynthesis approach works to find the best mixed implementation of HW and SW. It provides a systematic exploration of system designs that is driven by cost and performance constraints.

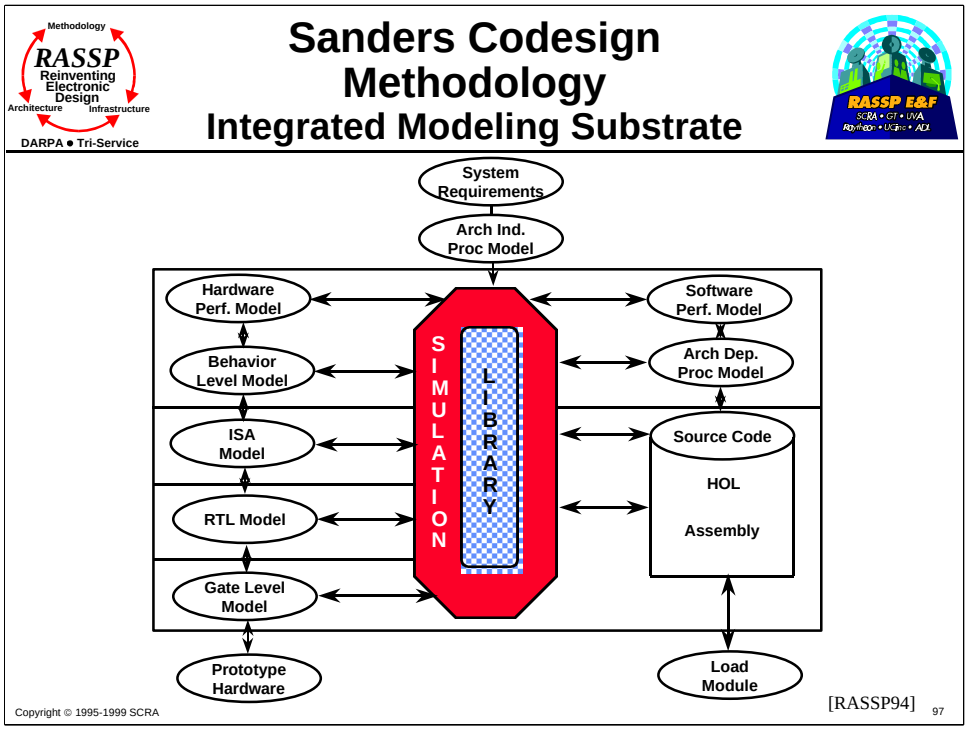
Module Outline

- 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



As part of the RASSP program, the prime contractors developed methodologies for codesign of embedded DSP systems. This slide shows the methodology developed by Lockheed Sanders. Their RASSP Development Methodology works top-down from requirements to completed system with feedback to the user at all stages and with an integrated HW/SW simulation in VHDL as a key.

[Hood94]

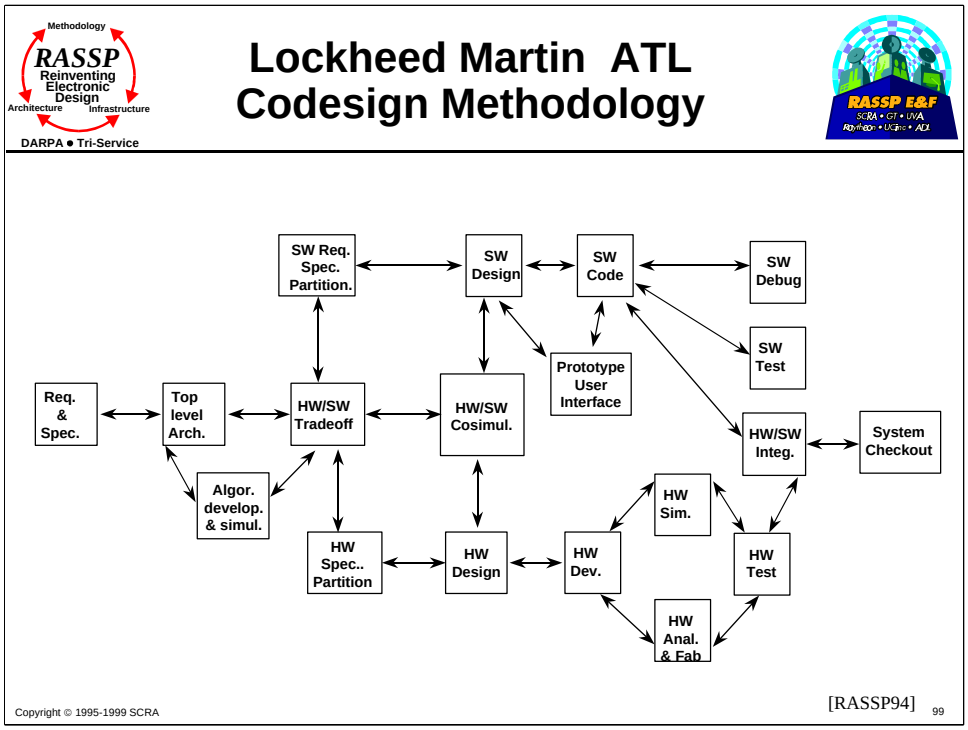


The Lockheed Sanders codesign methodology contains an attempt to construct an integrated modeling methodology to allow portions of the system, either hardware or software, to be described at different levels of abstraction.

Sanders Codesign Methodology

- **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**

Here are brief descriptions of the major processes in the Saunders codesign process.



This figure show the other RASSP prime contractor's (Lockheed Martin ATL) codesign process. Notice similarities to other methodologies:

- Requirements analysis
- Specification partitioning
- Tradeoff analysis
- Integration

Bi-directional arrows indicate that we may do incremental changes as needed in the design.

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

These are some of the major research efforts in codesign. A brief description of their major characteristics is included in the following slides and more details are included on the Chinook, Cosyma, Ptolemy, and Polis systems.

Chinook - [Chou95]

Cosmos - [Ismail95]

Cosyma - [Ernst93]

Polis - [Chiodo92]

Ptolemy - [Kalavade93]

Siera - [Srivastava91]

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*

The Chinook system is being developed by Gaetano Borriello's group at the University of Washington. The major area of research being looked at in Chinook is techniques for automating the synthesis of many different types of hardware/software interfaces. This includes automatic generation of device drivers on the software side, and glue logic on the hardware side.

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**

The Cosmos system is being developed at the National Poyltechnical Institute of Grenoble. The major emphasis is on the unified description in the Solar language, and the automated synthesis of communications channels between processes using 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

Cosyma is being developed at the University of Brunschweig. In Cosyma, the entire system is implemented in software running on embedded controllers. Functionality is migrated to hardware “accelerators” only if timing constraints are violated. Note that this is simply another type of partitioning heuristic.

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

POLIS is being developed by Alberto Sangiovanni-Vincentelli's group at UC Berkeley. The main areas of emphasis in POLIS are the use of CFSMs to provide an unbiased unified representation, and the automated synthesis of a hardware and software implementation once a suitable partition is found. The use of a unified representation that is very close to traditional FSMs also allows formal verification to be used.

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)**

Ptolemy is being developed by Edward Lee's group at UC Berkeley. The major area of emphasis and the original work in Ptolemy was the development of a heterogeneous simulation environment that allows the cosimulation of many different models of computation. This makes it ideally suited to modeling hardware/software systems.

RASSP Reinventing Electronic Design
Methodology
Architecture
Infrastructure
DARPA ● Tri-Service

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

Siera was developed by Robert Broderson's group at UC Berkeley. The major emphasis was the mapping of an application to a predefined architectural template and the integration of the tool with the UC Oct tools for implementation.

Chinook

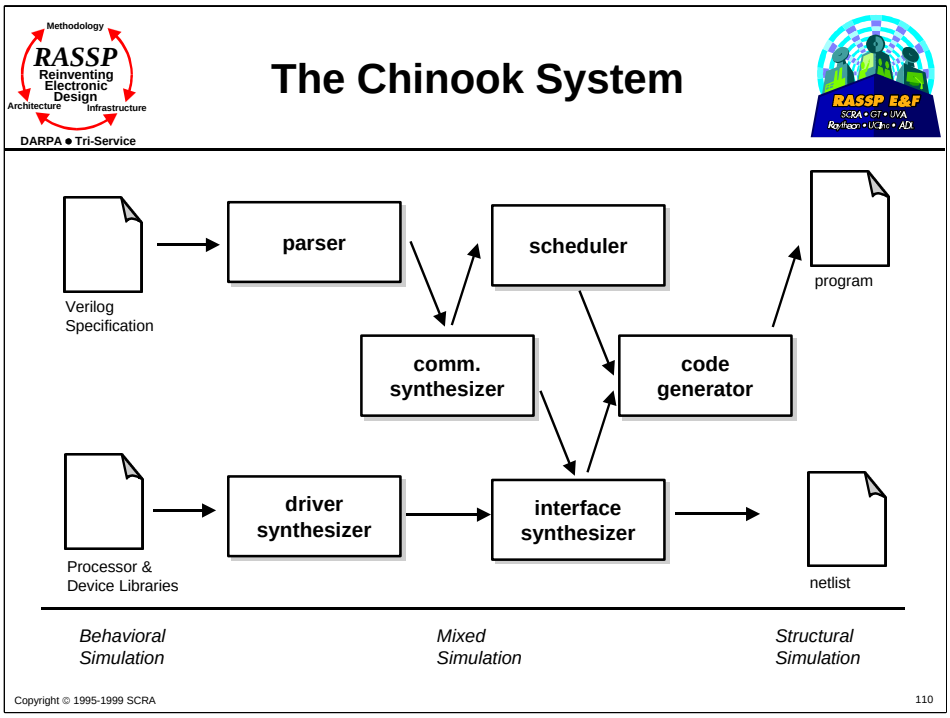
- Hardware/Software Co-synthesis system developed at the University of Washington
- Targeted at real-time reactive embedded systems
- Control dominated designs constructed from off-the-shelf components

This slide begins a more detailed look at the Chinook cosynthesis system. It is targeted towards the synthesis of control dominated systems to off-the-shelf processors and custom ASICs communicating through one of several different I/O mechanisms.

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

These are among Chinooks principal innovations. The claim is that the combination of them, not any individual one, is the most novel thing about Chinook.



This is the Chinook system. The inputs to the system are a Verilog behavioral specification of the system developed by the designer, complete with associated timing constraints, and a supplied library of processor and device libraries onto which to map the application.

The user must do the partitioning and mapping of the tasks onto processors or dedicated hardware by hand, but the system then automatically synthesizes all hardware and software for the system, including automatically synthesizing the interfaces between the processors and external hardware.

System Specification in Chinook
(Unified Representation)

- 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

Additional details on the Chinook system.

DARPA ● Tri-Service

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

Chinook provides an automated scheduling algorithm. Low level routines are grouped into modes, and then scheduled within them statically using heuristics to guide the search for a “good enough” schedule.

A customized dynamic scheduler is then generated for top-level modes if necessary.

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

Chinook automatically synthesizes interfaces between hardware and software portions of the system. Several different processor I/O styles including I/O ports and memory mapped I/O are handled for maximum flexibility in dealing with different processors.

Chinook synthesizes all of the additional software and hardware for the interfaces including device drivers and address/glue logic.

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)**

Chinook also can synthesize communications channels between tasks on different processors if a multiprocessor architecture is chosen.

Finally, Chinook includes an integrated modeling substrate based on the Verilog Programming Language. The Verilog PLI is used to interface between the software, written in C, and the hardware models, written in Verilog.

Cosynthesis of Embedded Applications (COSYMA)

- 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

This begins a section on more details concerning Cosyma. Like the other systems Cosyma is targeted towards a subset of hardware/software systems, in this case, small embedded real-time systems.

The partitioning strategy in Cosyma starts out with all tasks mapped to software. Then, when timing constraints are violated, tasks are migrated to hardware to attempt to generate a valid implementation that meets timing constraints.

[Hermann94].

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**

C* : a superset of the ANSI C standard.

Extensions of C:

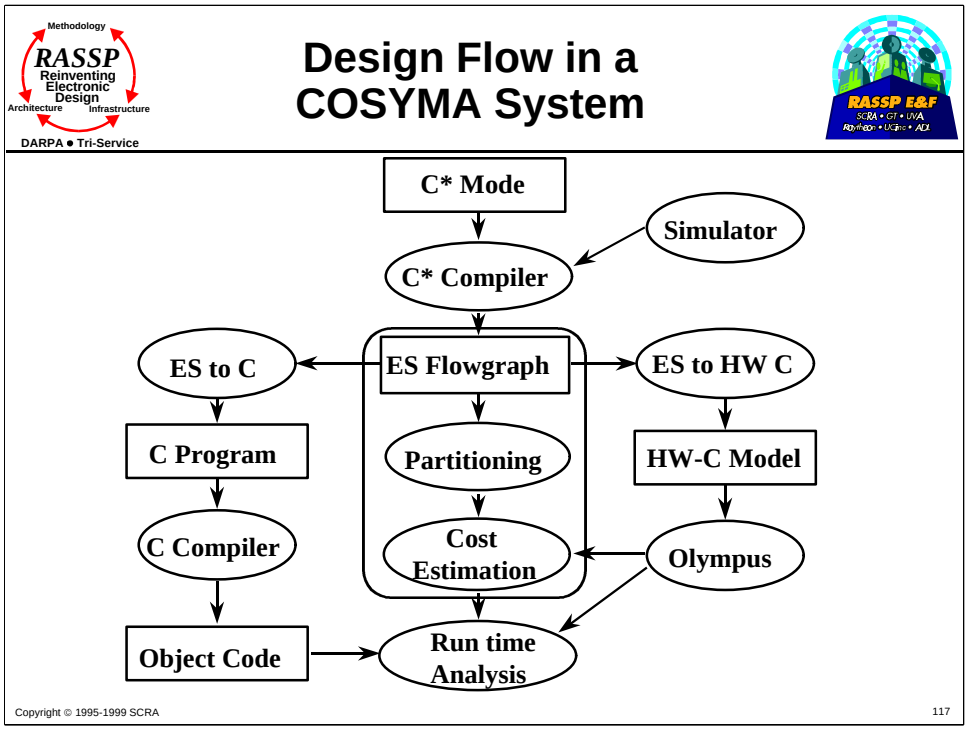
(1)Timing: min and max delays and duration between C labels of a task

(2)Task concept

(3)Task intercommunication

The COSYMA partitioning approach uses simulated annealing.

[Ernst93]



The input to Cosyma is a real time system described in a superset of the C language, C*, with time constraints and processes.

Input description is translated to an Extended Syntax Graph (ESG), which tries to compromise between differing requirements to the system.

1) HW/SW partition should take place at this level. This helps maintain independence from the hardware architecture.

2) Parts selected to be implemented in HW and SW, respectively, should be easily translated to their respective domains (HW or SW).

3) Dataflow analysis for different cosynthesis steps, like scheduling and translation to other target languages, should be supported.

[DeMicheli94], [Hermann94].

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 major emphasis for Cosyma is the development of automated partitioning algorithms, which few other systems attempt to implement.

As stated previously, the approach is to start with an all software solution and then move tasks to hardware to improve the schedule while minimizing cost. This multivariate optimization problem is handled using simulated annealing, a hill climbing algorithm.

[Hermann94].

COSYMA - Cost Function and Metrics

- 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

The Cosyma cost metric is based on estimates from the abstract, high level description. Using estimates of the metrics used in the cost computation rather than actual values leads to faster turnaround times.

[Hermann94].

COSYMA - Cost Function and Metrics (Cont.)

- 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

Details on the flow of the Cosyma tools after partitioning is performed.

[Hermann94].

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***

This slide gives a brief overview of Ptolemy. Ptolemy was started as a project to develop a heterogeneous simulation environment supporting many different models of computation. Although it was not specifically intended as a codesign environment, it is well suited to be the integrated modeling substrate for one. Some of the attributes that make Ptolemy well suited for HW/SW codesign include:

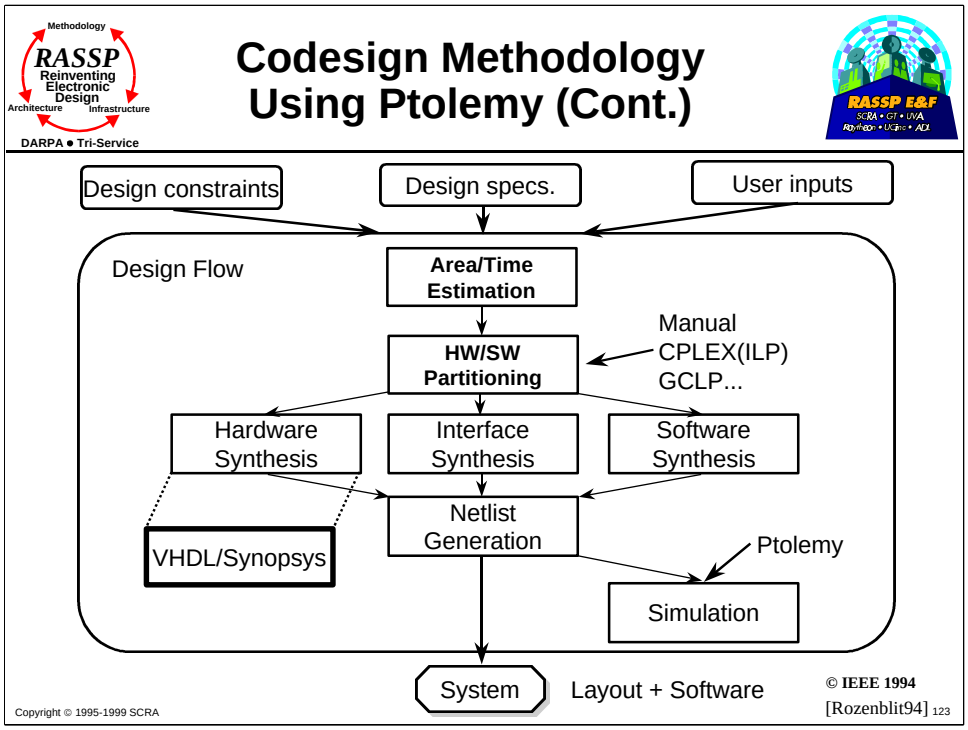
- Mixed-mode simulation and prototyping
- Block diagram description
- Model continuity

[Buck94]

Codesign Methodology Using Ptolemy

- **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**

An environment for codesign was added to Ptolemy. It was called the Design Assistant.



Another approach to codesign - here is the methodology used in conjunction with Ptolemy.

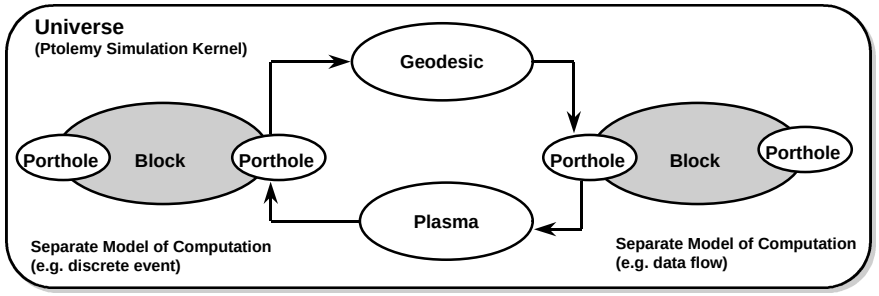
Note similarities to other codesign methodologies:

- Partitioning
- Iterative design at many stages of development.

Ptolemy approach supports simulation at various level of abstraction.

Ptolemy Heterogeneous Simulation Environment

Structural Components



- **Data encapsulated in "particles"**
- **"Block" objects send and receive messages**
- **Particles travel to/from external world through "portholes"**

This figure show some of the basic building blocks of Ptolemy The basic unit of modularity is a "block."

Portholes provide a standard interface through which blocks communicate.

Blocks communicate through streams called particles which form the base type for all messages passed.

Geodesic class establishes the connection between portholes.

Plasma class manages the reclamation of used particles."

A porthole is different from a "wormhole". A wormhole is a foreign subsystem contained entirely within an object.

[Kalavede, Lee, Ch. 19 from Rozenblit94]

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**

This slide begins a more detailed description of the POLIS system. POLIS is a codesign environment targeted toward small real-time reactive systems. The major features of POLIS include an "unbiased" specification mechanism and automated hardware and software synthesis after partitioning.

[Chiodo92], [Chiodo94]

POLIS
Unified Representation

- 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

The unified representation in POLIS is based on Codesign Finite State Machines (CFSMs). A CFSM is like an FSM except that all CFSMs are not defined to change state all at the same time (I.e., on a clock edge).

The system specification can be done in the ESTEREL language or in graphical extended FSMs, and this will be automatically translated into CFSMs. Once the entire system is specified in CFSMs, the system can be verified using formal verification techniques based on FSM theory. The system can then be partitioned by the designer with feedback in the form of cost estimates.

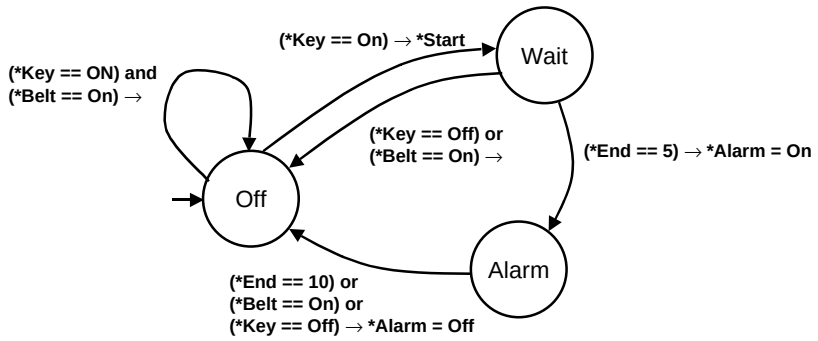
Once the system is verified and partitioned, the CFSMs that are to be implemented in hardware are translated into behavioral FSMs for hardware synthesis and the CFSMs that are to be implemented in software are translated into S-graphs for software synthesis.

Communication between CFSMs is through events. Events include a token (signal) and an associated value, if necessary. Normally, events between CFSMs are unidirectional and not buffered (I.e., the event sender can over write an event pending on one of its outputs). Events can be used to implement a fully interlocked handshaking protocol between CFSMs.

[Chiodo92], [Chiodo94]

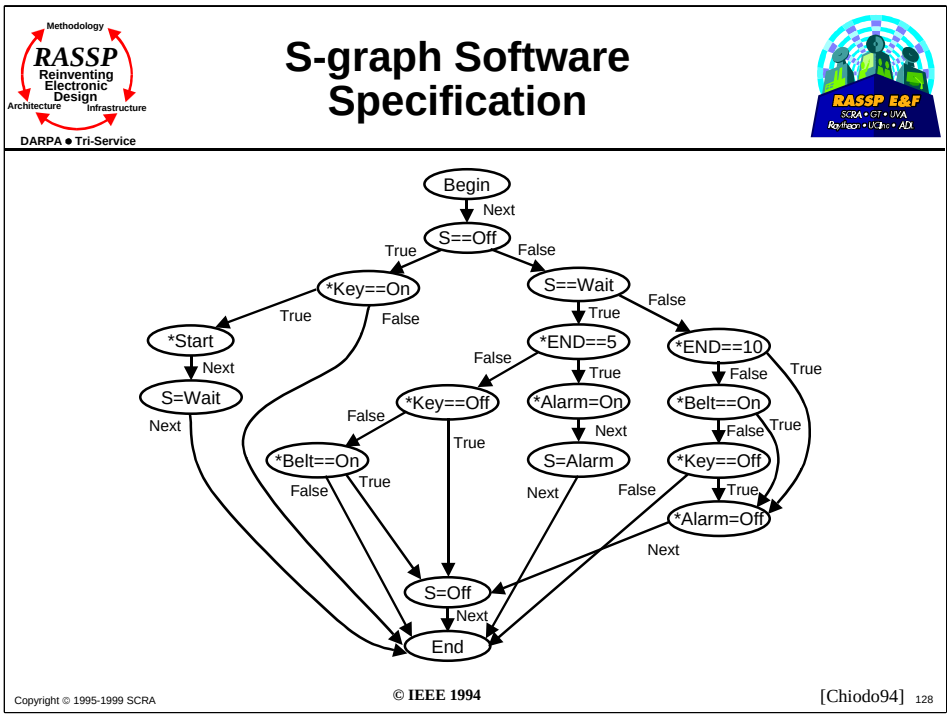
Codesign Finite State Machines

- **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"***



This is an example of a CFSM for a simple seat belt alarm system for a car.

[Chiodo94]



Here is the same CFSM as the previous slide translated into an S-graph. Note that the S-graph can now easily be coded into software.

[Chiodo94]

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

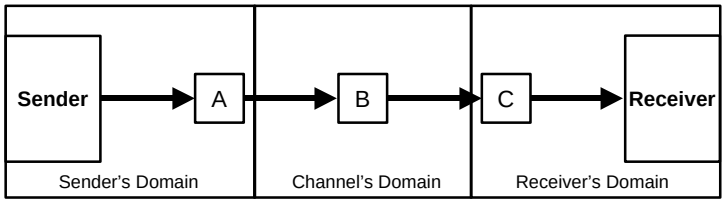
Once the user driven partitioning is completed, POLIS synthesizes the hardware, software, and interfaces between the two. There are seven interface styles available in POLIS depending on whether the interface is hardware-to-hardware, hardware-to-software, etc. and the communications mechanism used, I.e., asynchronous events or interlocked, etc.

Selection the scheduling algorithm to be used in the Real-Time Operating System (RTOS) in the embedded processors is left to the user.

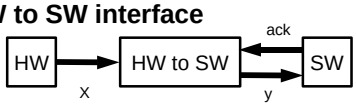
[Chiodo92], [Chiodo94]

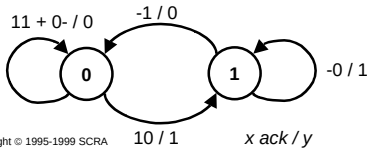
Interfaces Among Partitions

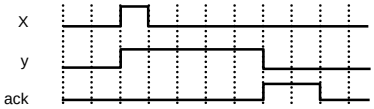
- **Interfaces use strobe/data protocol (corresponding to the event/value primitive)**



- **Example HW to SW interface**

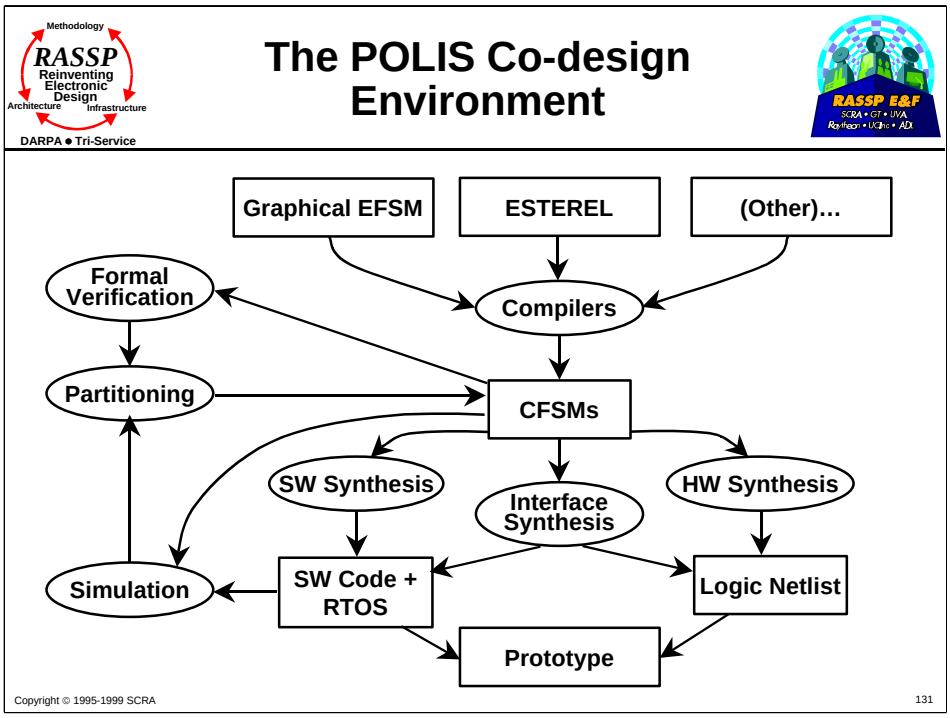






This slide show some more detail about how POLIS synthesizes the interfaces between partitions. The example on the bottom is the standard interface between hardware and software without interlocking.

[Chiodo92]



This is the POLIS codesign environment. As shown, the user specifies the system in ESTEREL, EFSM, or other suitable description languages, and these are automatically translated into CFSMs for formal verification, simulation, and partitioning. After partitioning, the system is automatically synthesized and then it can be simulated again at the implementation level.

[Chiodo92], [Chiodo94]

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**

RASSP Reinventing Electronic Design
Methodology
Architecture
Infrastructure
DARPA ● Tri-Service

Module Summary

RASSP E&F
SCRA • GT • UVA
Raytheon • Logic • ADI

- 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

Hardware/Software Codesign is becoming more and more necessary as mixed implementation systems become both more prevalent and more complex. This module has attempted to present some of the aspects of a good codesign environment and some of the research work being undertaken to develop one.

References

[Boehm73] Boehm, B.W. "Software and its Impact: A Quantitative Assessment," *Datamation,* May 1973, p. 48-59.

[Buchenrieder93] Buchenrieder, K., "Codesign and Concurrent Engineering", Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 85-86

[Buck94] Buck, J., et al., "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation*, Vol. 4, April 1994, pp. 155-182.

[Chiodo92] Chiod0, M., A. Sangiovanni-Vincentelli, "Design Methods for Reactive Real-time Systems Codesign," *International Workshop on Hardware/Software Codesign*, Estes Park, Colorado, September 1992.

[Chiodo94] Chiodo, M., P. Giusto, A. Jurecska, M. Marelli, H. C. Hsieh, A. Sangiovanni-Vincentelli, L. Lavagno, "Hardware-Software Codesign of Embedded Systems," *IEEE Micro*, August, 1994, pp. 26-36; © IEEE 1994.

[Chou95] P. Chou, R. Ortega, G. Borriello, "The Chinook hardware/software Co-design System," *Proceedings ISSS*, Cannes, France, 1995, pp. 22-27.

[DeMicheli93] De Micheli, G., "*Extending CAD Tools and Techniques*", Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 84

[DeMicheli94] De Micheli, G., "Computer-Aided Hardware-Software Codesign", *IEEE Micro*, August, 1994, pp. 10-16

[DeMichelli97] De Micheli, G., R. K. Gupta, "Hardware/Software Co-Design," *Proceedings of the IEEE*, Vol. 85, No. 3, March 1997, pp. 349-365.

[Ernst93] Ernst, R., J. Henkel, T. Benner, "*Hardware-Software Cosynthesis for Micro-controllers*", *IEEE Design and Test*, December, 1993, pp. 64-75

[Franke91] Franke, D.W., M.K. Purvis. "Hardware/Software Codesign: A Perspective," *Proceedings of the 13th International Conference on Software Engineering,* May 13-16, 1991, p. 344-352; © IEEE 1991

References (Cont.)

[Gajski94] Gajski, D. D., F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, Englewood Cliffs, N J, 07632, 1994

[Gupta92] Gupta, R.K., C.N. Coelho, Jr., G.D. Micheli. "Synthesis and Simulation of Digital Systems Containing Interactive Hardware and Software Components," *29th Design Automation Conference,* June 1992, p.225-230.

[Gupta93] Gupta, R.K., G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design and Test*, September 1993, p.29-40; © IEEE 1993.

[Hermann94] Hermann, D., J. Henkel, R. Ernst, "An approach to the estimation of adapted Cost Parameters in the COSYMA System*", 3rd International Conference on Hardware/Software codesign*, Grenoble, France, September 22-24, 1994, pp. 100-107

[Hood94] Hood, W., C. Myers, "RASSP: Viewpoint from a Prime Developer," *Proceedings 1st Annual RASSP Conference*, Aug. 1994.

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

[Ismail95] T. Ismail, A. Jerraya, "Synthesis Steps and Design Models for Codesign," *IEEE Computer*, no. 2, pp. 44-52, Feb 1995.

[Kalavade93] A. Kalavade, E. Lee, "A Hardware-Software Co-design Methodology for DSP Applications," *IEEE Design and Test*, vol. 10, no. 3, pp. 16-28, Sept. 1993.

[Klenke96] Klenke, R. H., J. H. Aylor, R. Hillson, D. J. Kaplan, "VHDL-Based Performance Modeling for the Processing Graph Method Tool (PGMT) Environment," *Proceedings of the VHDL International Users Forum*, Spring 1996, pp. 69-73.

[Kumar95] Kumar, S., "A Unified Representation for Hardware/Software Codesign", Doctoral Dissertation, Department of Electrical Engineering, University of Virginia, May, 1995

[Jalote91] Jalote, P., *An Integrated Approach to Software Engineering,* Springer-Verlag, New York, 1991.

[McFarland90] McFarland, M.C., A.C. Parker, R. Camposano. "The High-Level Synthesis of Digital Systems," *Proceedings of the IEEE*, Vol. 78, No. 2, February 1990, p.301-318, © IEEE 1990.

References (Cont.)

[Parker84] Parker, A.C., "Automated Synthesis of Digital Systems," *IEEE Design and Test*,, November 1984, p. 75-81.

[RASSP94] *Proceedings of the 1st RASSP Conference*, Aug. 15-18, 1994.

[Rozenblit94] Rozenblit, J. and K. Buchenrieder (editors). Codesign Computer -Aided Software/Hardware Engineering, IEEE Press, Piscataway, NJ, 1994; © IEEE 1994.

[Smith86] Smith, C.U., R.R. Gross. "Technology Transfer between VLSI Design and Software Engineering: CAD Tools and Design Methodologies," *Proceedings of the IEEE*, Vol. 74, No. 6, June 1986, p.875-885.

[Srivastava91] M. B. Srivastava, R. W. Broderson, "Rapid prototyping of Hardware and Software in a Unified Framework," *Proceedings ICCAD*, 1991, pp. 152-155.

[Subrahmanyam93] Subrahmanyam, P. A., "Hardware-Software Codesign -- Cautious optimism for the future", Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 84

[Tanenbaum87] Tanenbaum, A.S., *Operating Systems: Design and Implementation,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1987.

[Terry90] Terry, C. "Concurrent Hardware and Software Design Benefits Embedded Systems," *EDN*, July 1990, p. 148-154.

[Thimbleby88] Thimbleby, H. "Delaying Commitment," *IEEE Software,* Vol. 5, No. 3, May 1988, p. 78-86.

[Thomas93] Thomas, D.E., J.K. Adams, H. Schmitt, "A Model and Methodology for Hardware-Software Codesign," *IEEE Design and Test,* September 1993, p.6-15; © IEEE 1993.

[Turn78] Turn, R., "Hardware-Software Tradeoffs in Reliable Software Development," *11th Annual Asilomar Conference on Circuits, Systems, and Computers,* 1978, p.282-288.

[Vahid94] Vahid, F., J. Gong, D. D. Gajski, "*A Binary Constraint Search Algorithm for Minimizing Hardware During Hardware/Software Partitioning",* 3rd International Conference on Hardware/Software Codesign, Grenoble, France, Sepetember22-24, 1994, pp. 214-219

[Wolf94] Wolf, W.H. "Hardware-Software Codesign of Embedded Systems," *Proceedings of the IEEE*, Vol. 82, No.7, July 1994, p.965-989.

References (Cont.)

Additional Reading:

Aylor, J.H. et al., "The Integration of Performance and Functional Modeling in VHDL" in *Performance and Fault Modeling with VHDL,* J. Schoen, ed., Prentice-Hall, Englewood Cliffs, N.J., 1992.

D'Ambrosio, J. G., X. Hu, "Configuration-level Hardware-Software Partitioning for Real-time Embedded Systems", *3rd International Conference on Hardware/Software codesign*, Grenoble, France, September 22-24, 1994, pp. 34-41

Eles, P., Z. Peng, A. Doboli, "VHDL System-Level Specification and Partitioning in a Hardware-Software Cosynthesis Environment", *3rd International Conference on Hardware/Software codesign*, Grenoble, France, September 22-24, 1994, pp. 49-55

Gupta, R.K., G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design and Test*, September 1993, p.29-40.

Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997

Schultz, S.E., "An Overview of System Design," *ASIC and EDA*, January 1993, p.12-21.

Thomas, D. E, J. K. Adams, H. Schmit, "A Model and Methodology for Hardware-Software Codesign", *IEEE Design and Test*, September, 1993, pp. 6-15

Zurcher, F.W., B. Randell, "Iterative Multi-level Modeling - A Methodology for Computer System Design," *Proceedings IFIP Congress '68,* Edinburgh, Scotland, August 1968, p.867-871.