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

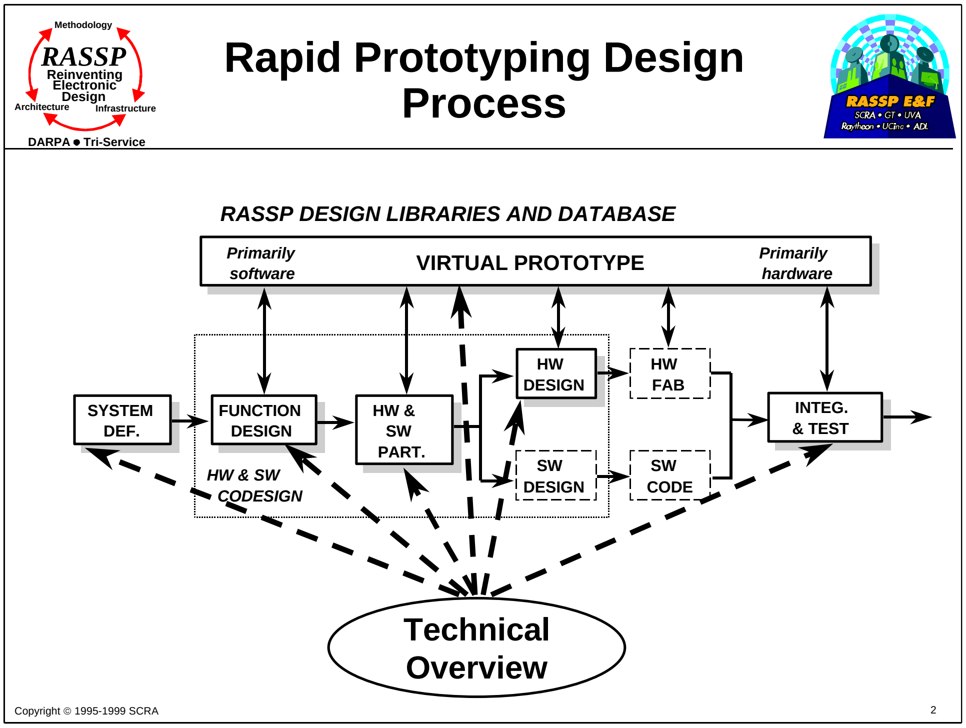
RASSP Methodology Overview

RASSP Education & Facilitation Program

Module 29

Version 3.00

The successful Rapid Prototyping of Application-Specific Signal Processors (RASSP) program of the US Department of Defense (DARPA and Tri-Services) targets a 4X improvement in the design, prototyping, manufacturing, and support processes (relative to current practice). We present a recent industrial system design practice model and the RASSP methodology for the design and prototyping of application-specific signal processors developed as part of the DARPA's RASSP Education & Facilitation (E&F) Program. A number of limitations in current design practice are highlighted together with a number of candidate RASSP solutions, and some of the future challenges.



The RASSP Methodology and Design Flow is shown in this slide. The underlying basis of the RASSP design methodology is the technology of virtual prototyping. Virtual prototyping begins with the early requirements description and ends with the detailed test and fielding of the system.

We may also emphasize the role of VHDL in the RASSP program. VHDL can be used for system definition, functional design, hardware-software partitioning, hardware design and hardware-software integration and test. The concept of virtual prototyping uses VHDL as the binding language of choice for all design paradigms.

The most common usage of VHDL prior to RASSP was in the area of hardware design. The RASSP program has extended VHDL's use to include executable requirements, performance modeling/system level design as well as system integration and test. Many of these developments have led to the proposal for a System-Level Design Language (SLDL) in the late 90s.

Module Goals

- **Identify problems with current design methodologies**
- **Introduce the RASSP design methodology and its solution to current practices**
- **Introduce the concept of an evolving virtual prototype**

 3

A number of problems existed in current practice in digital systems design in the early 1990s. As part of the RASSP effort, these problems were identified and a new approach centered on the virtual prototyping process had been proposed as a cost effective and efficient methodology for system-level design.

This module attempts to describe the problems that were endemic to system-level design, and then discusses the RASSP approach.

RASSP Methodology Overview Outline

- **Overview**
  - **RASSP Technical Approach**
- **Methodology**
  - **Overview**
    - **Current Design Practices**
    - **The RASSP Approach**
  - **Virtual Prototyping**
    - **Executable Requirements/Specifications**
      - **Description**
      - **Case Study**
    - **Data/Control Flow Modeling**
      - **Description**
      - **Case Study**

After presenting an overview of current methodology and RASSP design flow, we will present the various phases in the RASSP design methodology, ranging from requirements to detailed design.

RASSP Methodology Overview Outline

- Cost Modeling
  - Description
  - Case Study
- Performance Modeling
  - Description
  - Case Study
- Fully Functional Modeling
  - Description
  - Case Study
- Model Year Architecture
- Reuse

- **Results to Date**
- **Summary**

 5

We end the module with a summary of main results obtained in the program, and references to RASSP publications.

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

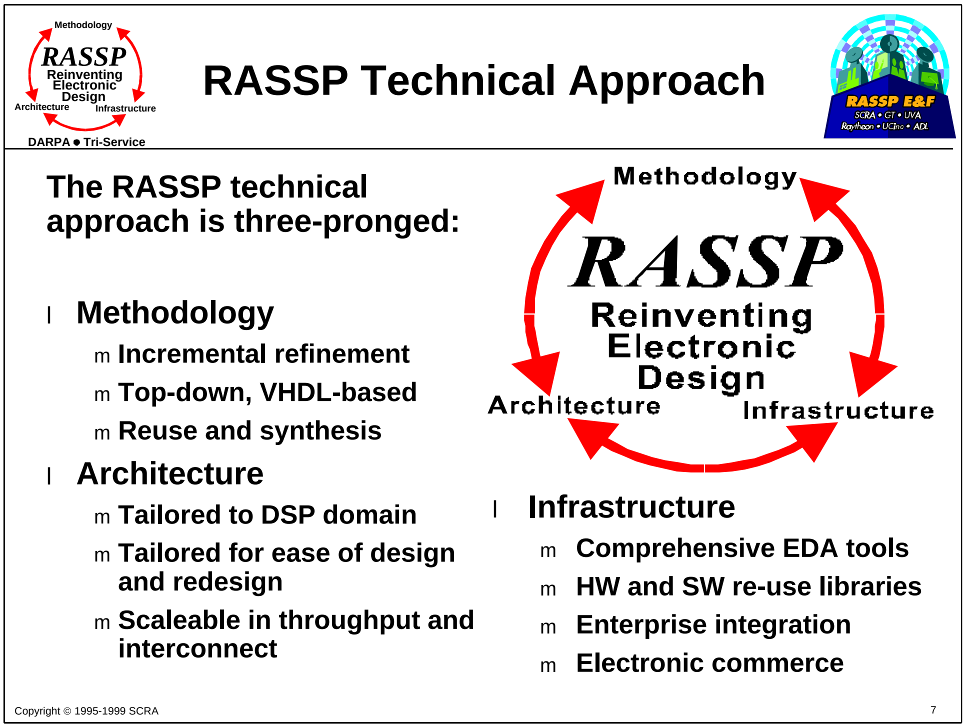
RASSP Methodology Overview Outline

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Overview**
- **Methodology**
  - **Overview**
  - **Virtual Prototyping**
  - **Model Year Architecture**
  - **Reuse**
- **Results to Date**
- **Summary**

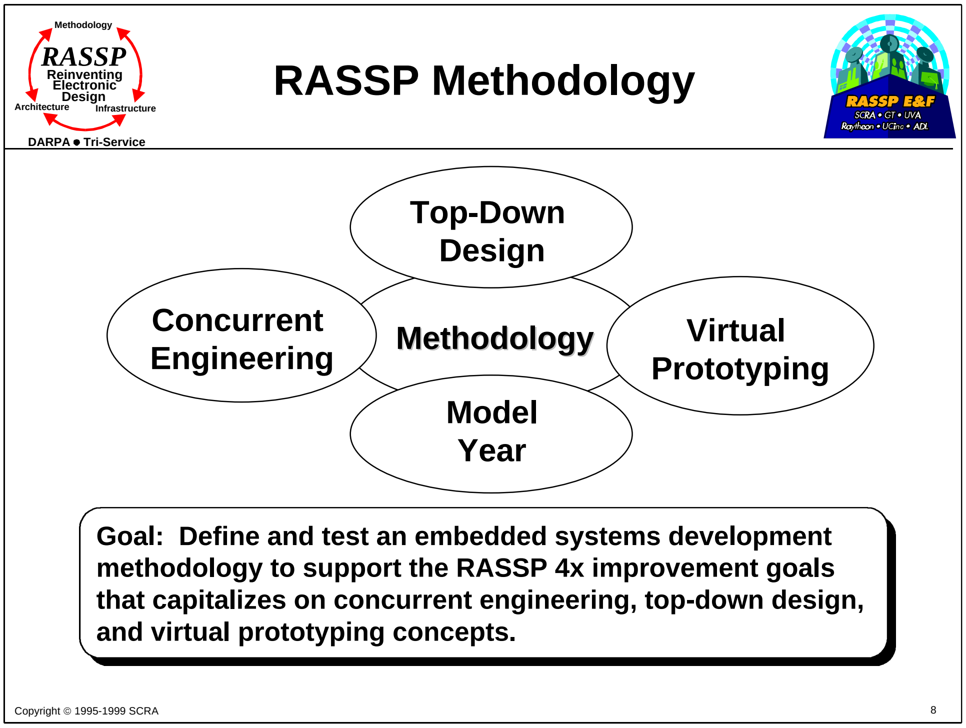
 6

The overview presents a snapshot of RASSP design methodology in terms of its fundamental technology thrusts.

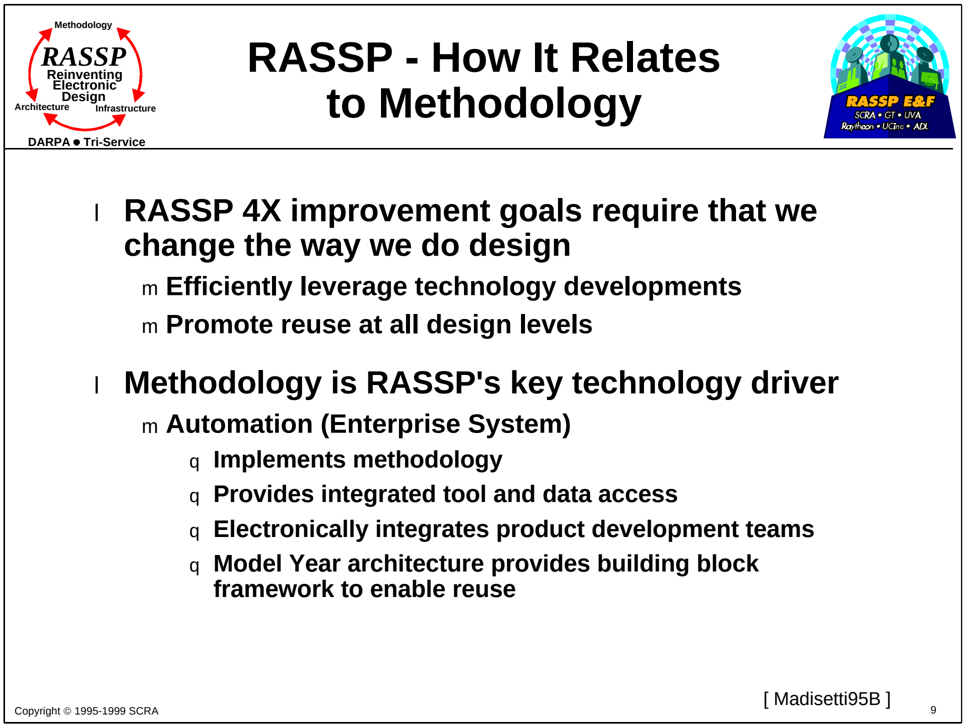


The RASSP technical approach rests on the three pillars of Methodology, Architectural innovation, and Infrastructure support. The Methodology relies on a top down methodology that iteratively and incrementally develops a design through various stages of abstraction. Cost is minimized and design time is reduced through reuse of previous stored information. The Architectural effort relies on architectures that are application-specific, and thus can be quickly tailored to a particular mission within that general application, reducing risk and cost. In the area of Infrastructure, these are extensive support for frameworks of EDA tools and database libraries of models of COTS components.

If each of the above factors resulted in 5-10 % improvement over current practice (independently of others), then the overall improvement can be truly significant, leading to a promise of 4X improvement in cost and quality.



The main pillars of RASSP methodology are highlighted. The notion of Virtual Prototyping will be expounded in detail in later slides. Concurrent engineering (that involves concurrent coordinated interaction & activity between design and product teams in the very least) is well understood in the industrial community. Model Year approach favors incremental and iterative improvement of an existing design over several iterations (or model years), similar to the use of the term in the automotive industry.



Methodology should be considered independent of tools, in that tools realize an implementation of the proposed methodology.

RASSP - How it Relates to Methodology (Cont.)

- Methodology defines key technology extensions
  - Emphasizes concurrency and design reuse
  - Links system requirements capture to architecture tradeoffs
  - Defines library-based design approach
  - Verifies virtual prototype design at all levels before manufacture

[Madisetti95B]

All along the process, the virtual prototype is having detail added to it.

The prototype includes:

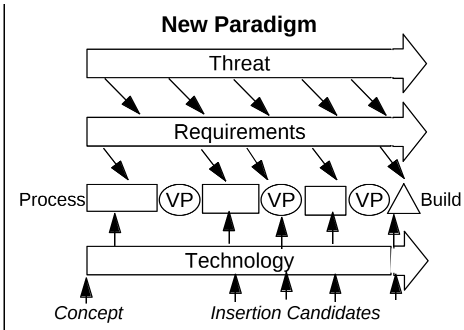
Requirements tracking

Hardware description

Software description

Test patterns

RASSP Methodology

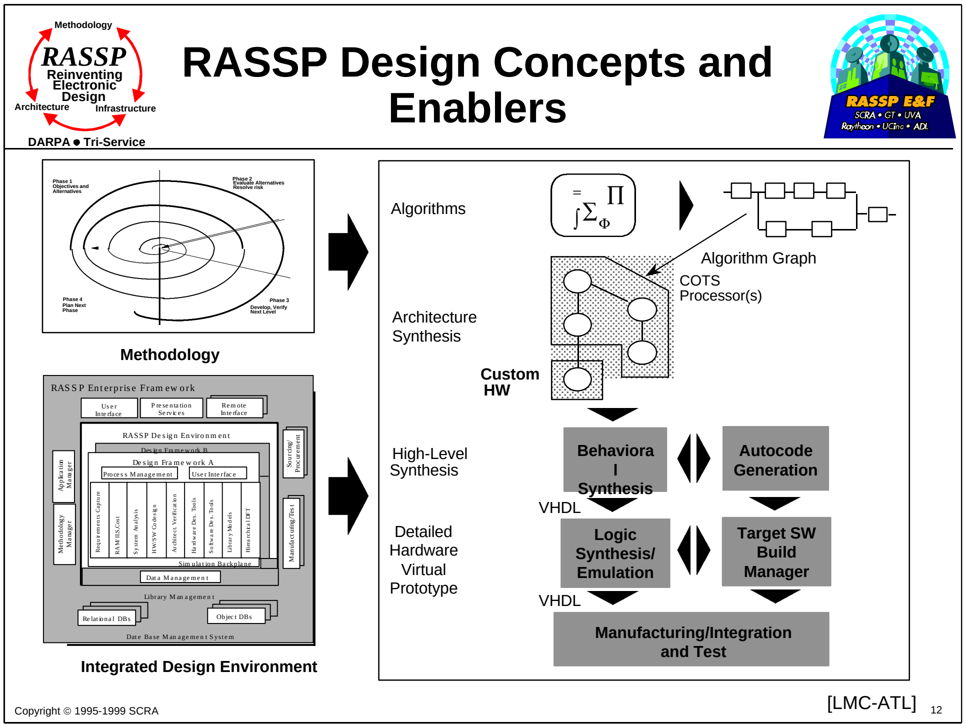


- **Traditional designs**
  - — Static, sequential process (waterfall model)
  - — Custom designs
  - — Technology dated when fielded
  - — High design (NRE) and life cycle costs (LCC)

- **RASSP Virtual Prototyping (VP)**
  - — Dynamic, risk-driven concurrent process (spiral model)
  - — Incorporates evolving requirements
  - — Rapid insertion of COTS technology
  - — State-of-the-art fielded product
  - — Low cost insertion, LCC

[LMC-ATL]

- The major methodology change being pursued under RASSP is occurring in how requirements are mapped to implementation approaches and verified via virtual prototyping. The challenge of implementing this methodology is to evolve modeling and simulation tools and models that support the hierarchical design and verification process. This concept supports deploying the latest technology design via a virtual prototype that is easily mappable to a manufacturable approach.



• The Lockheed Martin ATL RASSP approach is pursuing a spiral model concept that is supported by an integrated Enterprise environment that supports design tools for use at all levels of the design hierarchy.

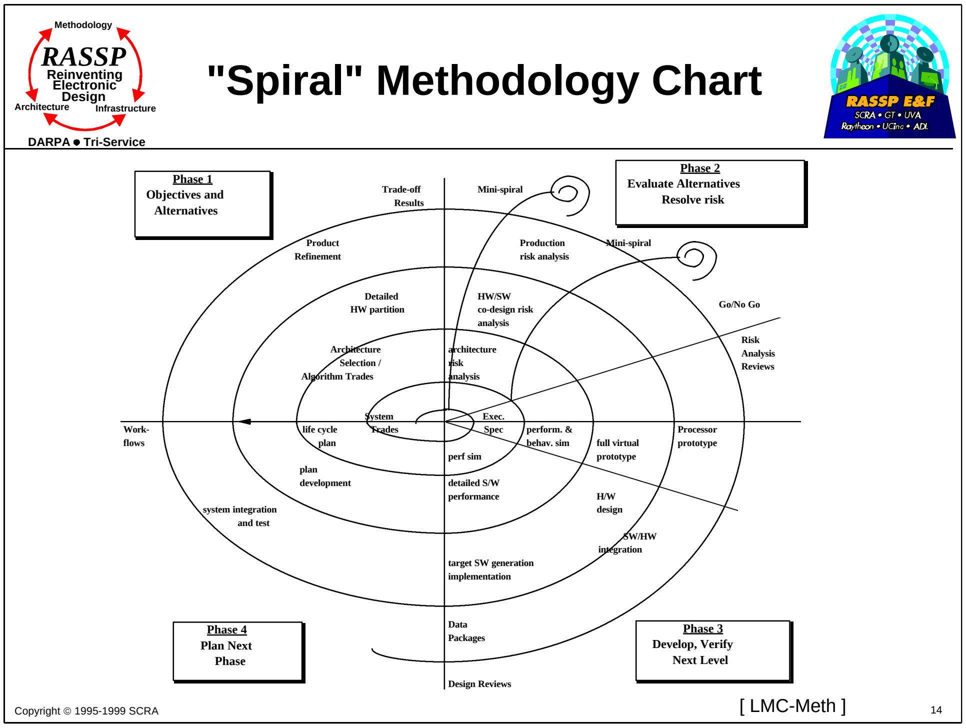
• Specific emphasis is being placed on developing a methodology and tool set that allows starting with algorithms and mapping to a codesign trade-off approach that supports developing hardware and software approaches that support manufacturing and integration.

Codesign, Virtual Prototyping, and Design Reuse Enables the $4X^+$ Goal

Iterative Virtual Prototype - the "Spiral" Model

- **The main purpose of this model is to make improvements over the traditional "waterfall" model. It emphasizes:**
  - **Objectives and alternatives studies**
  - **Risk analysis reviews**
  - **Development/design reviews**
  - **Reevaluation of work-flows**
- **As the design develops, it spirals through these reviews**
- **Each iteration of the spiral results in a more sophisticated version**
- **This data package then drives the next iteration of the design**

A spiral model allows iterative improvement of the design with feedback between the various stages of an evolving design. The spiral design thus minimizes risk by considering a variety of alternatives along each axis before incrementing the current version of the design.



Four phases are associated with each major cycle of the spiral.

Phase 1: The baseline approach and appropriate alternatives are developed to meet program objectives.

Phase 2: The approaches are evaluated against the objectives and alternatives, and the risks associated with these approaches are evaluated.

Phase 3: The prototype is evaluated, and the next level of the product is developed. This phase results in a prototype of the design.

Phase 4: The product is reviewed, and plans for the next development stage are established.

The entire process is then repeated to the next level of detail.

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

RASSP Methodology Overview Outline

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Overview
- **Methodology**
  - Overview
  - Virtual Prototyping
  - Model Year Architecture
  - Reuse
- Results to Date
- Summary

 15

We will now present the RASSP methodology.

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

Detailed Section Outline

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Overview**
  - **Current Design Practices**
  - **The RASSP Approach**
- **Virtual Prototyping**
  - **Executable Requirements/Specifications**
    - **Description**
    - **Case Study**
  - **Data/Control Flow Modeling**
    - **Description**
    - **Case Study**
  - **Cost Modeling**
    - **Description**
    - **Case Study**
  - **Performance Modeling**
    - **Description**
    - **Case Study**

16

A typical high-performance avionics parallel signal processor operation flow consists of three stages - sensor signal processing (SSP), application-specific signal processing (ASP), and mission-specific signal processing (MSP). The inputs are recorded by sensor arrays, and the data is pre-processed by an array of (typically hardwired) computational elements, comprising the sensor-specific processing (SSP), that are optimized with the sensor array and the recording environment. Typical SSP operations include range adjustment, background subtraction,and matched filtering.Given the high computational throughput and restricted functionality, and severe form constraints (size, volume, area and power), the SSP functions are typically ASICs with non-standard interfaces. SSP functions are also referred to as time-dependent processing. After this time-critical processing is completed, the application-specific(ASP) parallel processing (about 30-100 processors) is commenced on an array.. (continued on next slide).

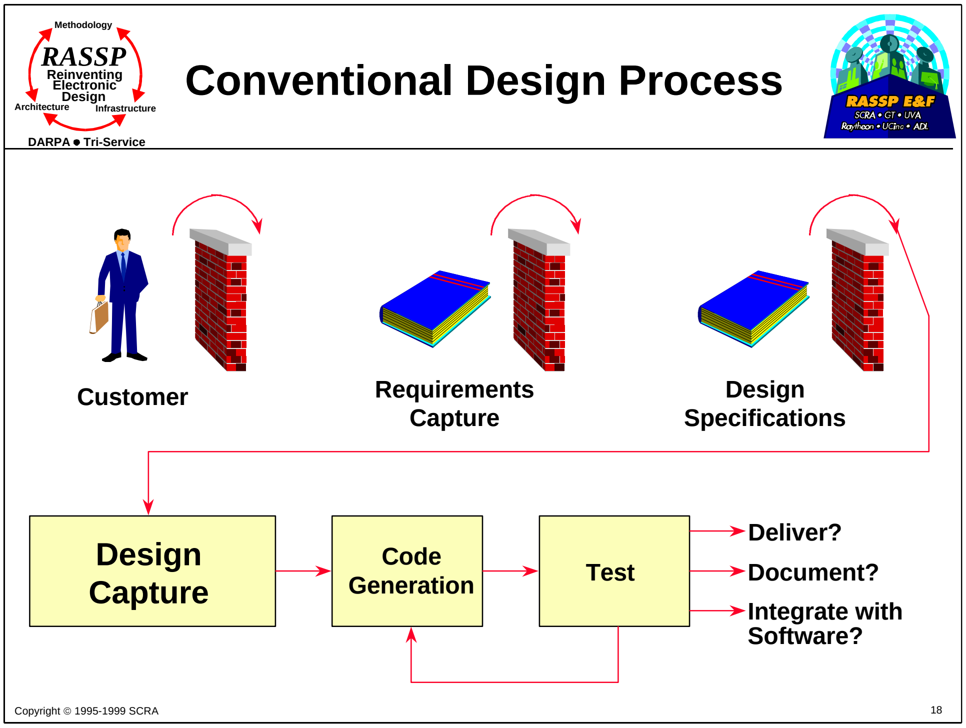
Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA • Tri-Service

Detailed Section Outline (Cont.)

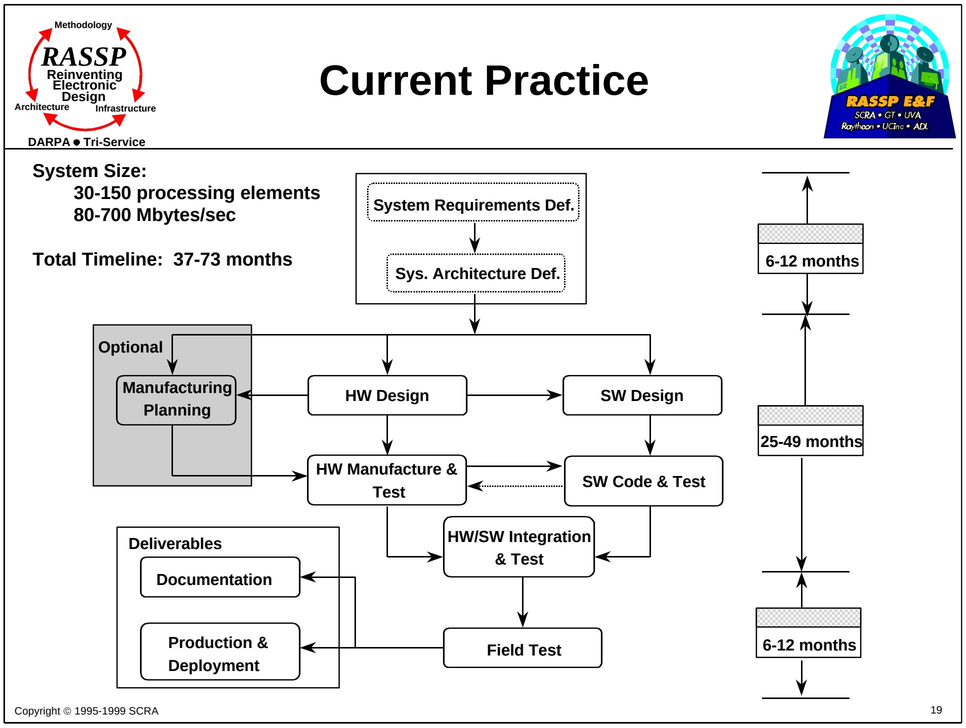
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Fully Functional Modeling
  - Description
  - Case Study
- **Model Year Architecture**
- **Reuse**

of processors and communications elements, with appropriate test, control, and maintenance structures. Typical ASP operations include coordinate transformation, track-to-track correlation, Kalman filtering, tracking, and parametric estimation and involve application related functionality. The ASP functions also require relatively high throughput, and it is desired that they have as much flexibility (i.e., programmability) as possible, together with certain form factors. In an ASP implementation lie the multi-objective function optimization and tradeoffs among form factors, performance, programmability,ease of upgrades, and capability for test and diagnostics. ASP functions can also be referred to as object-dependent processing.The mission-specific (MSP) processing typically requires interpretation of the ASP processing, and can be confined to a few processors that are often co-located within the ASP box. These functions include clutter analysis, track handoff, decision analysis, kill assessment, etc. Typical form factor constraints for volume, power,weight, and I/O rates are in the order of 2-10 cuf, 40-500W, 10-60 lbs, and 4-30 Mbytes/second, while for low-end low power portable applications they are considerably more severe (in size and power). Inter-processor communication bandwidth requirements can range between 40-1000 Mbytes/second.



In the conventional system design process (Circa 1993-1994), the customer requirements are not captured in a systematic manner or in an executable form. These, often very vague requirements are converted to design specifications, usually in an ad hoc and manual fashion. This is followed by conversion of the design specifications into an executable form, followed by code generation (for hardware synthesis and software design) and test. Many issues are left unverified or vague in this process, leading to lengthy design verification cycles and errors in requirements, specifications, and test. Furthermore, different teams are assigned to each of the intermediate steps leading to futher inefficiency in the design process.



A current practice model is required as a baseline to help assess the improvements afforded by the RASSP process.

The focus of the RASSP program is on signal processors consisting of a few to hundreds of processing elements.

This diagram shows the time frames related to current practice broken down into various phases of development. These include:

- •Architecture Analysis (6 to 12 months)
- •HW and SW Design along with integration (25 to 49 months)
- •Field prototyping and test (6 to 12 months)

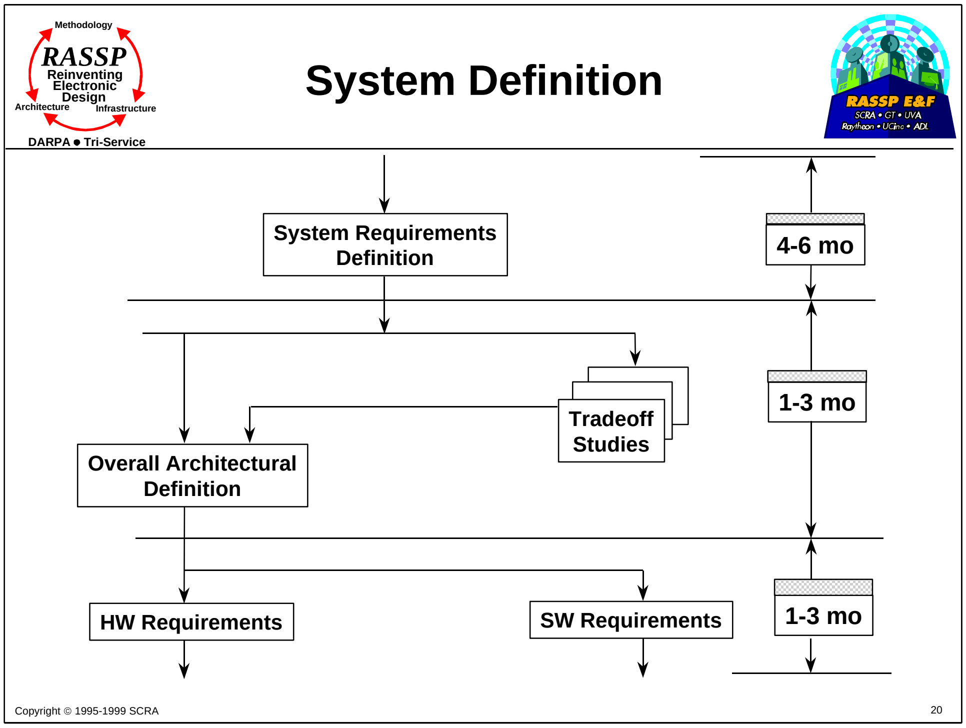
These will be decomposed further in the following slides.

This chart follows a waterfall approach to design methodology which is typical of current practice circa 1993.

The underlying concept of the waterfall process is a progression through various levels of abstraction, or phases, with the intent of fully characterizing each level before moving to the next.

The following bad design practices tend to result from this process:

- •Limited use of concurrent engineering
- •Solving wrong problems early in design process
- •Inflexibility late in design process
- •Significant rework and cost resulting from design flaws found late in the process

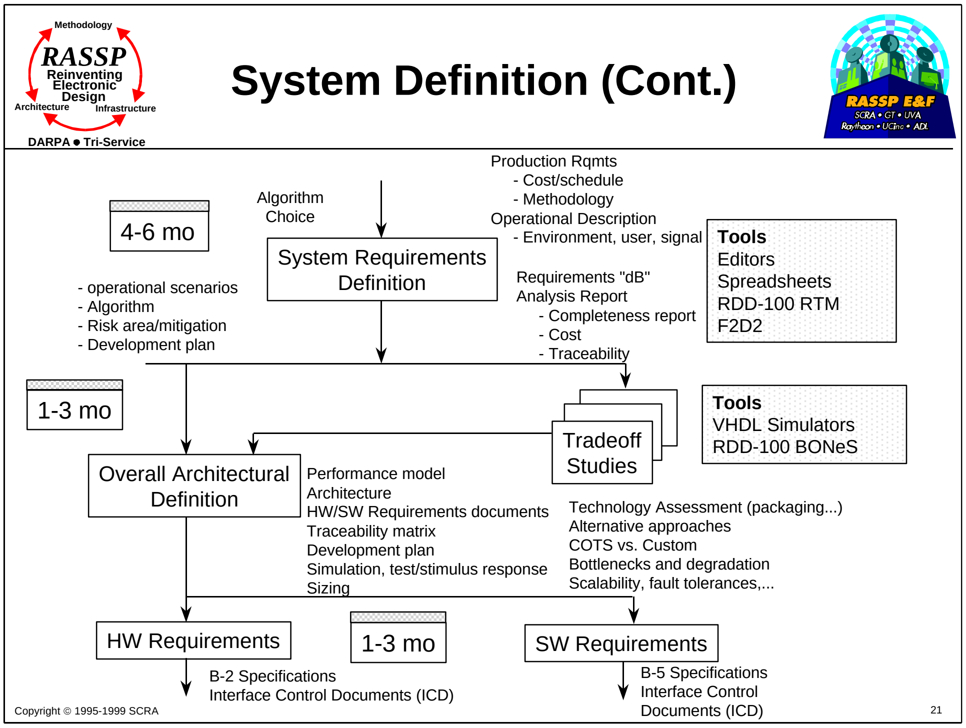


Current practice is to provide processor and system requirements in written form, often in hundreds of pages of documentation. The total time for this step is 6 to 12 months.

To improve this time, tools are needed which automate the decomposition of information down to the next level of design. Also, tools are needed which can do trade-off analysis at each level of design.

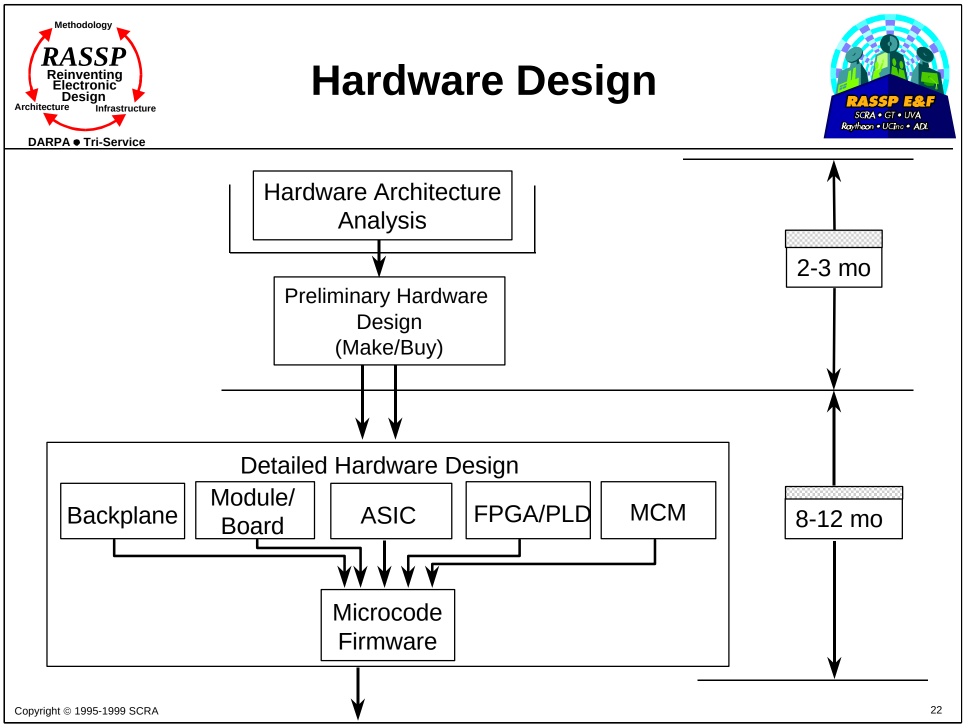
System requirements must be converted into design functionality

- Functionality must be linked so that it can be traced back to the system requirements that it is meeting
- Tests must be established which fulfill the requirements and are linked back to the system requirements which they validate.



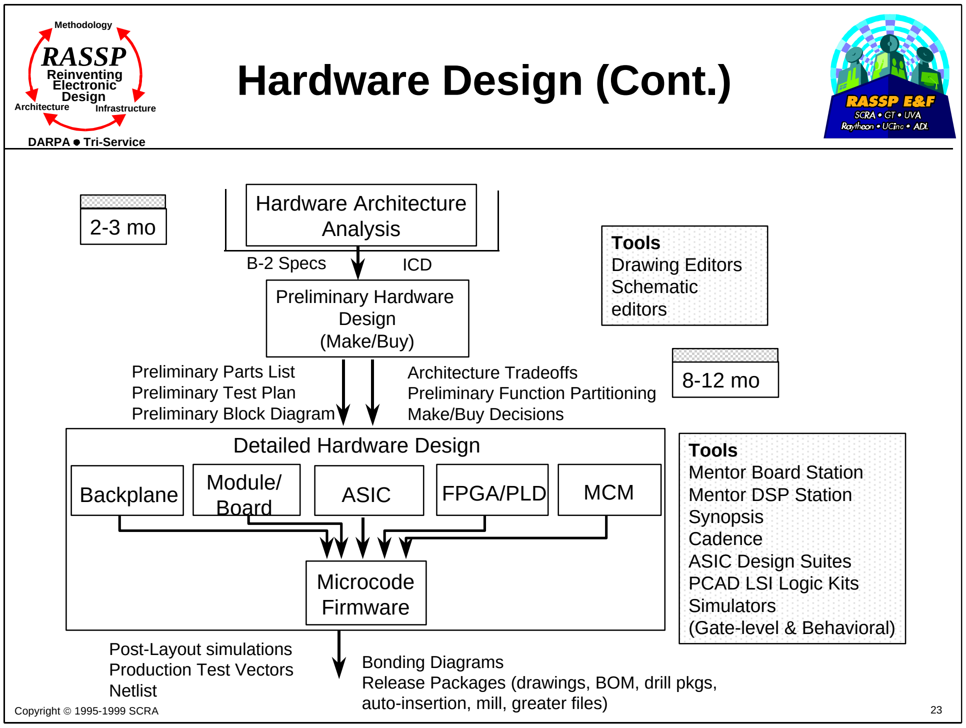
Some tools are being developed and refined to handle the system requirement flow. "Executable requirements and specifications" would put the requirements into a machine-readable and -executable form.

RDD-100 is a tool which captures requirements.



This "waterfall" chart depicts hardware and firmware development. The entire process currently takes from 10 to 15 months.

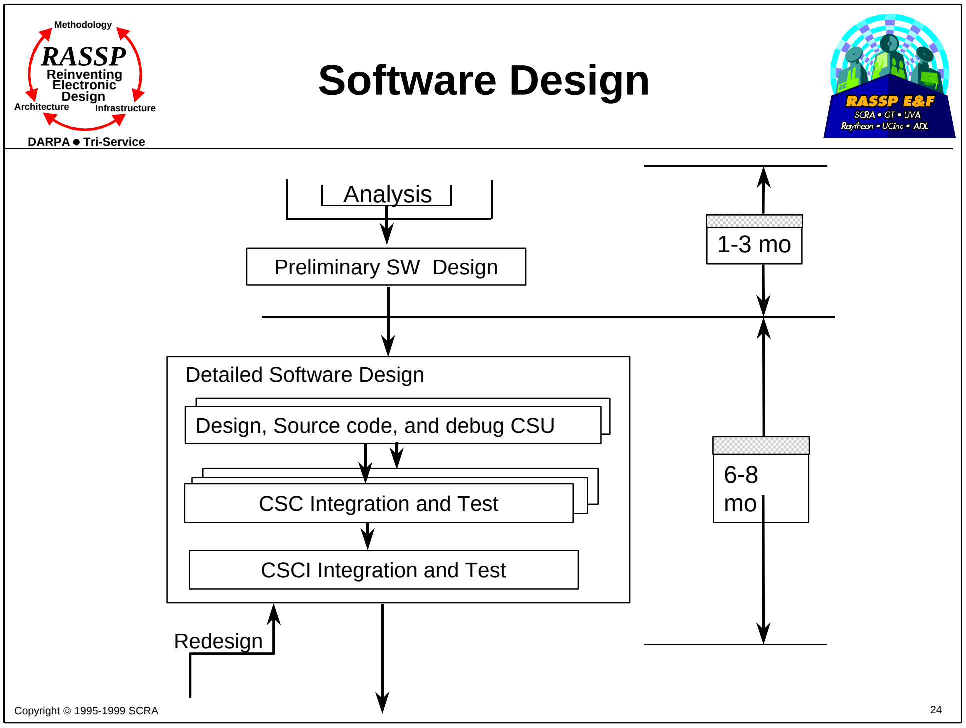
This has long been recognized as an area where computer simulation and layout tools can be applied. There are many tools on the market which speed up these engineering-intensive processes.



Modeling tools, which create a model that can be expanded in detail at next-lower levels, are now emerging. VHDL-based systems help meet this requirement.

Recently, there have been efforts by some vendors to develop a common product data description database that can be used to represent the design at all levels of the design process.

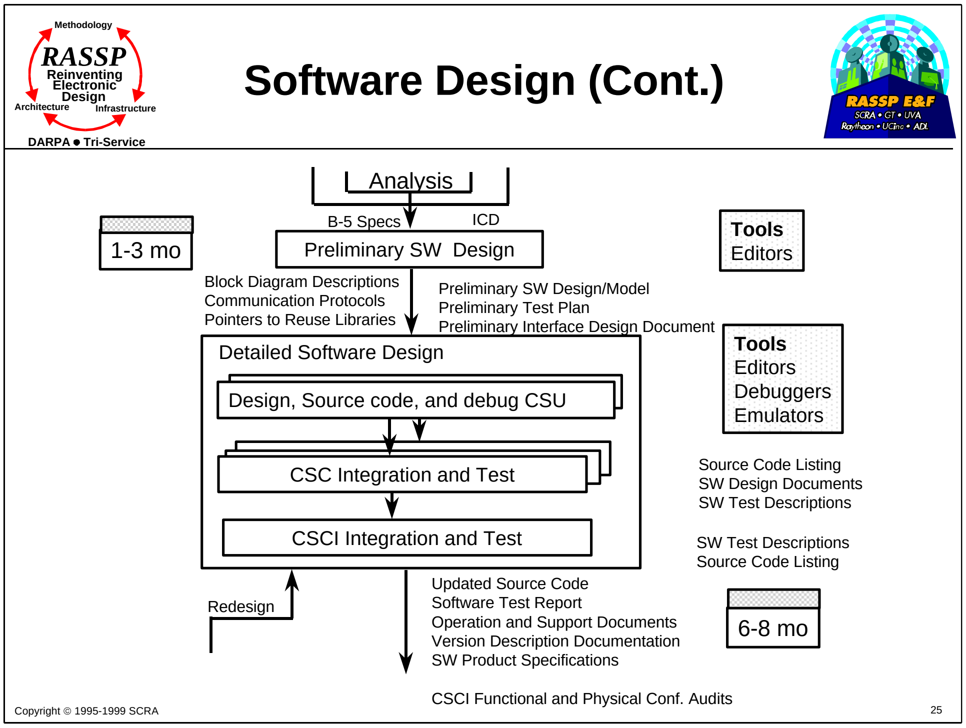
Layout tools are also being developed which track layout effects and feed them back to the simulation so that designers can verify that system requirements are being met.



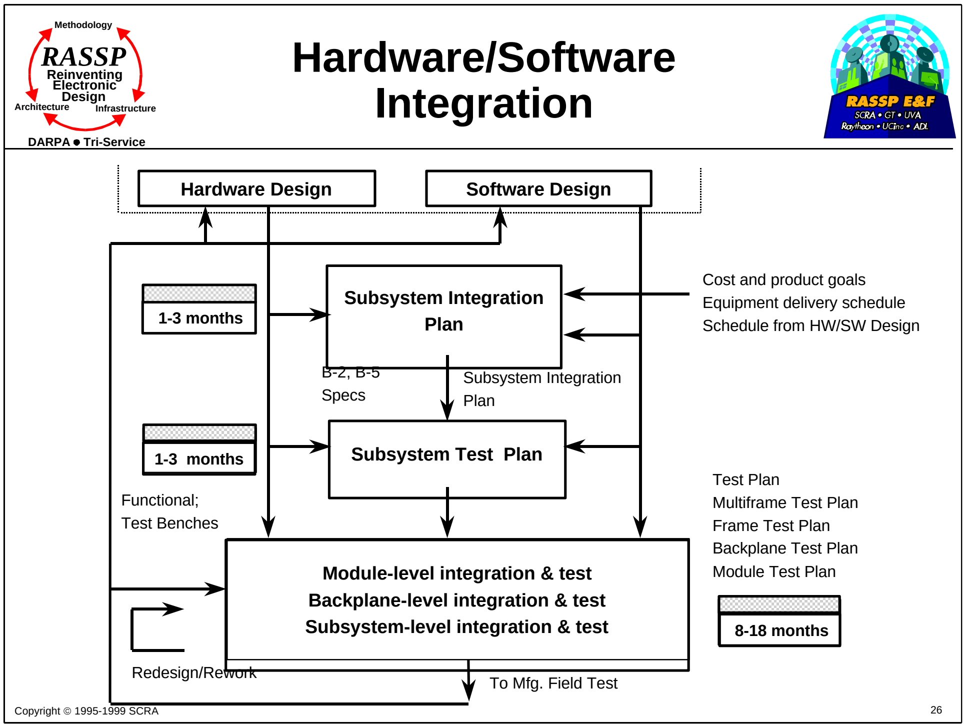
The software design area typically takes from 7 to 11 months, and the issues of integration can extend on for months more.

Problems:

Tradeoffs between hardware and software implementations are hard to evaluate. System requirements are not easily represented or traceable through software code. In-process changes to the system requirements are not easily propagated to the system code.



Software design is probably the best understood process in the system design methodology, and the typical design flow is described above.



Integration occurs when the HW and most of the SW are ready. A plan for integration must be created to guarantee sufficient coverage of the HW and SW.

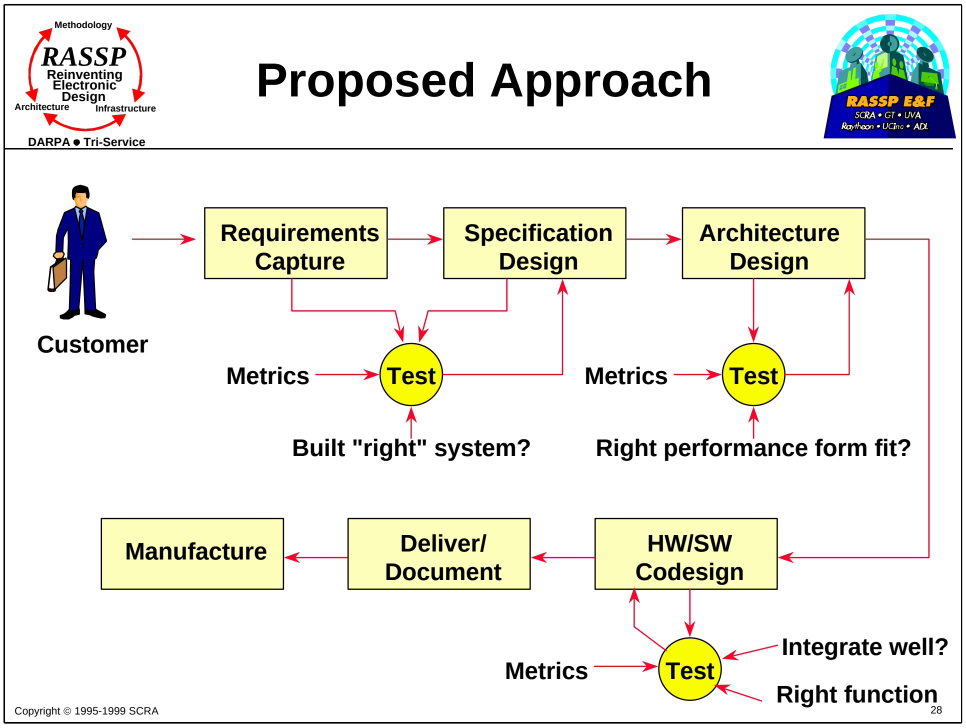
The actual integration and test can take from 8 to 18 months depending on the number of design flaws and SW work-arounds required.

Detailed Section Outline

- Overview
  - Current Design Practices
  - **The RASSP Approach**
- Virtual Prototyping
  - Executable Requirements/Specifications
    - Description
    - Case Study
  - Data/Control Flow Modeling
    - Description
    - Case Study
  - Cost Modeling
    - Description
    - Case Study
  - Performance Modeling
    - Description
    - Case Study

 27

We now describe the RASSP approach.



To mitigate the risk involved in requirements and specifications capture and derivation, RASSP puts emphasis on these early tasks to ensure that the requirements and specifications are captured in an executable form together with test benches to ensure early and rapid verification. The focus is on building the right system, with the right architectures, and correct detailed design through the use of hierarchical verification.

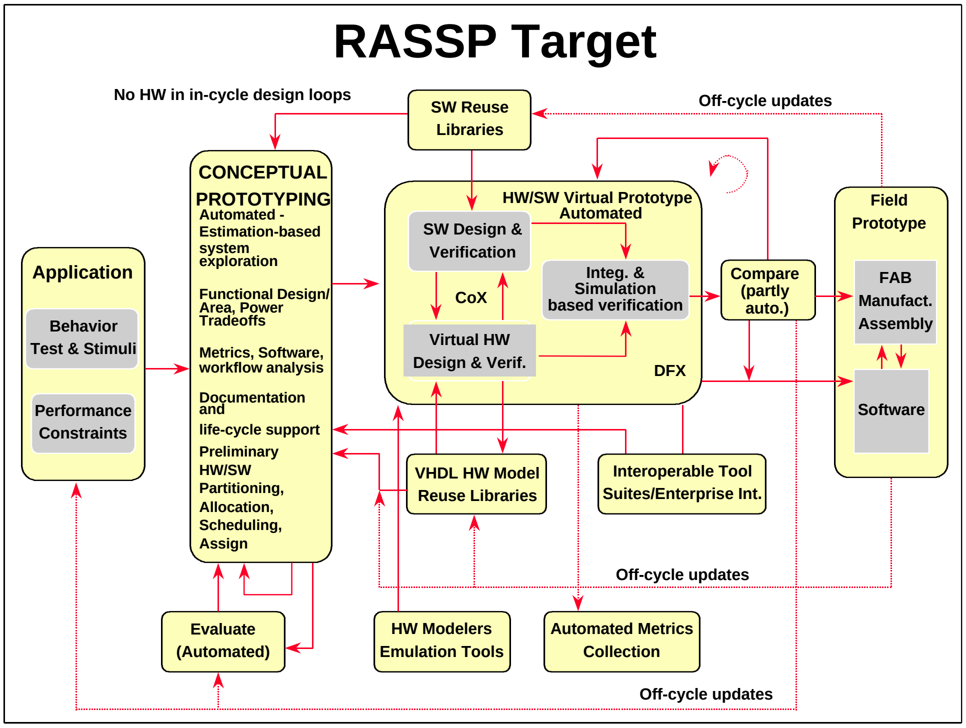
RASSP Reinventing Electronic Design
Methodology · Architecture · Infrastructure
DARPA ● Tri-Service

Rationale for New Process

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

• **An industry survey of design practices highlights the importance of the Requirements Capture and the Specifications phases in the design process.**
• **The requirements phase and the integration/documentation phases contribute to 70% of the design effort and 40% of the possible errors that arise in a typical system design.**
• **On the average, a typical organization removes only 82% of the possible errors in a delivered product, while top organizations remove 95% of the errors in the delivered product.**
• **Most of the these delivered defects arise from requirements ambiguity and lack of a formal process for system level design.**

1995 Software Productivity Research Inc. survey



In the RASSP design flow, an early stage, defined as "conceptual prototyping" which involves early design, and replaces the manual HW/SW partitioning block of the "current practice". Conceptual prototyping utilizes automated tools that allow rapid estimation and evaluation of algorithmic, functional, architectural and enterprise-related trade-offs early in the design process. A few candidate conceptual prototypes are then culled from the dozen or so generated at this stage, and then passed on to the virtual prototyping stage. Here, extensive evaluation and detailed design is done in virtual hardware and software leading to successful and rapid integration, again through the use of HW/SW reuse libraries, interoperable tools and enterprise integration. The entire process depends heavily on automation, and feedback currently being obtained from benchmark designs on candidate RASSP-like processes by the primes and other RASSP participants will be used to refine and improve upon both the rapidity, as well as the correctness of the first-time prototyping efforts of large DSP systems. The envisioned process presents a number of open problems related to both conceptual and virtual prototyping and verification that are to be effectively addressed by various RASSP and the larger electronic systems design and application community, promising an exciting time for digital system designers trying to cut the prototyping times by a factor of four. (See V. Madisetti, "Vive La Difference," The RASSP Digest, Vol 1, 4th Quarter 1994).

Features and Limitations of Existing Codesign Methodologies

DSP Codesign Features	Thomas/ Adams '93	Kumar/ Aylor '93	Gupta/ De Micheli '93	Kalavade/ Lee '93 & '94	Ismail/ Jerraya '95	RASSP Method
Executable Functional Specification	✓	✓	✓	✓	✓	✓
Executable Timing Specification		✓	✓			✓
Automated Architecture Selection						✓
Automated Partitioning			✓	✓		✓
Model-based Performance Estimation		✓	✓			✓
Economic Cost/Profit Estimation Models						✓
HW/SW Cosimulation				✓		✓
Uses IEEE Standard Languages		✓				✓
Integrated Test Bench Generation						✓

RASSP differs from current practice in many ways:

1. No hardware fabrication, assembly, and test is present in in-cycle design loops.

2. Late binding of hardware allows the design product to be state-of-shelf at time of manufacture or use.

3. Extensive use of conceptual and virtual prototyping optimizes efficiency of the final product, and guarantees right-first time designs.

4.Design reuse supported by generation, maintenance, and upgrades of application-specific VHDL libraries for rapid design of signal processors.

5. Enterprise integration and interoperability between various point design tools facilitates design portability and standardization.

6. Extensive use of automation to facilitate --- a nested-loop and iterative design process, automated metrics collection and distributed collaboration facilities for large design project management speeds up the prototyping, a documentation and life-cycle maintenance process. (For further details on this table see V. Madisetti and J. DeBardelaben, "A RASSP Approach to HW/SW Codesign, The RASSP Digest, Vol. 2, 4th Quarter, 1995).

Detailed Section Outline

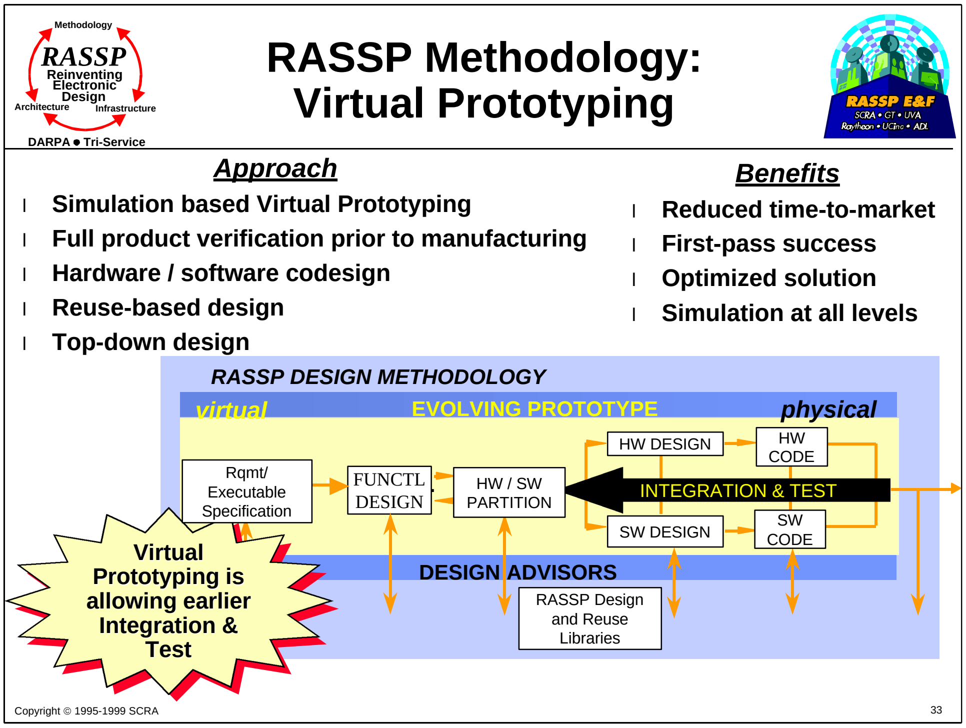
- Overview
  - Current Design Practices
  - The RASSP Approach
- **Virtual Prototyping**
  - Executable Requirements/Specifications
    - Description
    - Case Study
  - Data/Control Flow Modeling
    - Description
    - Case Study
  - Cost Modeling
    - Description
    - Case Study
  - Performance Modeling
    - Description
    - Case Study

 32

A virtual prototype is a computer simulation model of a final product, component, or system. Unlike the other modeling terms that distinguish models based on their characteristics, the term virtual-prototype does not refer to any particular model characteristic but rather it refers to the role of the model within a design process; specifically for the role of: exploring design alternatives, demonstrating design concepts, testing for requirements satisfaction/correctness.

Virtual prototypes can be constructed at any level of abstraction and may include a mixture of levels. Several virtual prototypes of a system under design may exist as long as each fulfills the role of a prototype. To be useful in a larger system design, a virtual-prototype model should define the interfaces of the component or system under design.

In contrast to a physical prototype, which requires detailed hardware and software design, a virtual prototype can be configured more quickly and cost-effectively, can be more abstract, and can be invoked earlier in the design process. A distinction is that a virtual prototype, being a computer simulation, provides greater non-invasive observability of internal states than is normally practical from physical prototypes (See RASSP Taxonomy Document).

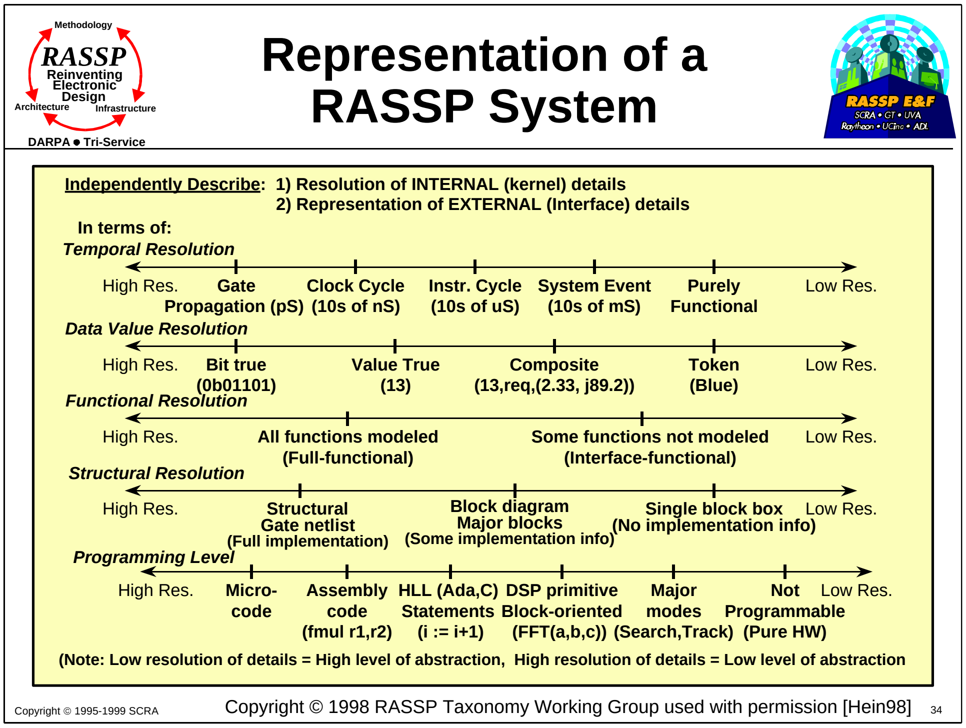


The ability of designers to rapidly develop and field application–specific signal processing is dependent on their ability to accurately model the systems that they wish to build. This modeling starts with modeling of the application to be developed, and it continues through architectural analyses and into detailed design.

Accurate modeling of the system being developed is key to good selection of architecture and to rapid development of hardware. Good models of hardware speed development by reducing errors in design and by allowing simultaneous hardware/software development.

The modeling begins with a functional description and proceeds through a series of refinements to produce detailed hardware and software. During this refinement process, as a sequence of models are developed to model system function, system performance, system detailed behavior, and detailed system design. The use of a common modeling allows the system engineers, the hardware engineers, and the software engineers all to interact on a common, executable, model of the processing problem.

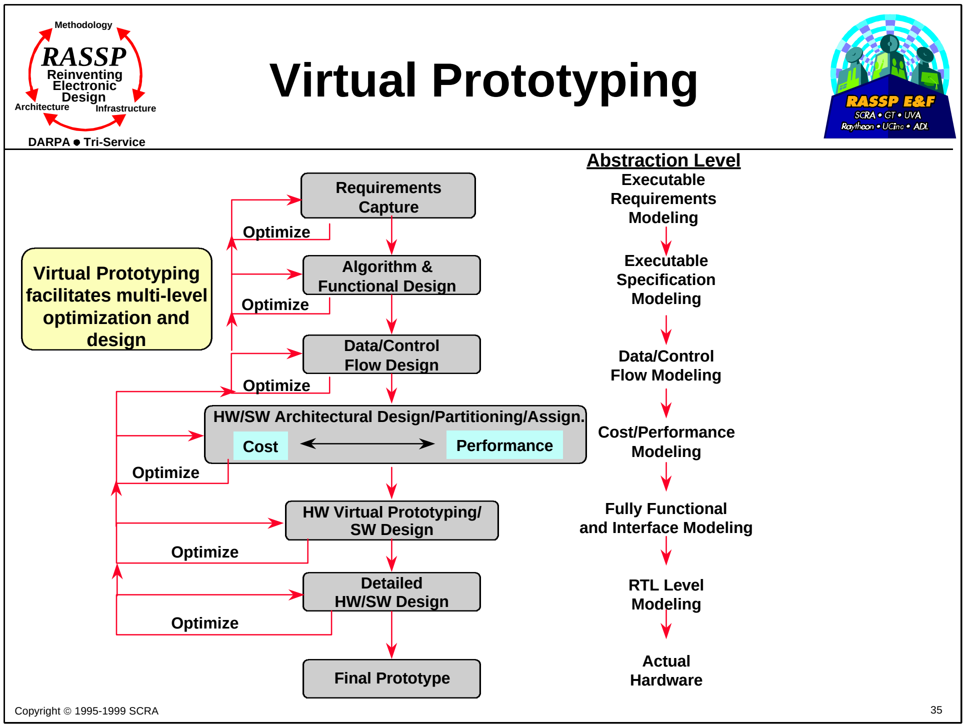
The development of a complete system model with the proper structure allows the development team to catch and eliminate several hardware interface errors that would normally have been found after physical integration (See Madisetti & Egolf, IEEE Micro, Fall 1995, pp. 9-21).



The taxonomy represents model attributes that are relevant to designers and model users. It is based on common terminology that is readily understood and used by designers. The taxonomy consists of a set of attributes or axes that characterize a model's relative resolution of details for important model aspects.The taxonomy axes, shown in the slide, identify five distinct model characteristics:

1. Temporal detail
2. Data Value detail
3. Functional detail
4. Structural detail
5. Programming level

The first four attributes do not completely address the hardware/software codesign aspect of a model, because they do not describe how a hardware model appears to software. The fifth axis represents the level of software programmability of a hardware model or, conversely, the abstraction level of a software component in terms of the complementary hardware model that will interpret it (See RASSP Taxonomy Document).



This slide describes an outline of the virtual prototyping process where the requirements are converted to specifications followed by other levels of the design abstraction. The feedback loops support spiral design flow with test and verification (through simulation) of each decision made in the virtual prototyping process.

Each of the levels is supported by its library of models, and design exploration and verification is supported in a hardware-less simulation based environment.

For futher details on the design flow shown, please see: T. Egolf, "VHDL-Based Rapid System Prototyping," *Journal of VLSI Signal Processing*, Vol. 14, Issue 2, 1996.

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

Definitions

RASSP E&F — SCRA • GT • UVA — Raytheon • UCinc • ADL

- **Virtual prototype**
  - **An executable requirement or specification of an embedded system and its stimuli that describes it in operation at multiple levels of abstraction.**
- **Virtual prototyping**
  - **A top down design process of creating a virtual prototype for hardware and software cospecification, codesign, cosimulation, and coverification of the embedded system.**
- **Conceptual prototyping**
  - **A software representation of a component, board/MCM, or system that supports early architectural design, modeling and performance evaluation of candidate architectures including manufacturing and life cycle cost functions.**
- **Hardware/Software virtual prototype**
  - **A software representation of a hardware component, board/MCM, or system integrated with its application, diagnostic, and control code containing sufficient accuracy to guarantee its successful realization.**

 36

The RASSP design methodology is derived from the traditional top down design paradigm with the incorporation of the Virtual Prototype concept. The basis of this concept is to develop a complete description, in standard languages like VHDL, C, Java, and Ada, prior to fabrication. The design is checked out completely as a model prior to commitment to hardware. In this way design errors are caught when they are easy to fix, and the system performance can be validated in simulation.

The choice of VHDL as the modeling language is important because VHDL provides:

Completeness: VHDL provides the mechanism for capturing the system behavior in a form that can be maintained and upgraded for twenty years or more; and

Portability: VHDL is an industry standard so models developed in VHDL can be ported to a wide range of simulation environments and can be maintained over the system's lifetime.

Virtual Prototyping Should Provide the Following

- "Represent" the prototype during various stages in system development process
- "Represent" the prototype at multiple levels of abstraction in top-down design
- Allow optimization of design at multiple levels or different stages
- "Document" the design for effective upgrades and support
- Be cost effective (time and dollars)

One must understand the attributes of the system being designed to be able to "represent" it accurately
"Represent" = Modeling

 37

The ability of designers to rapidly develop and field application–specific signal processing is dependent on their ability to accurately model the systems that they wish to build. This modeling starts with modeling of the application to be developed, and it continues through architectural analyses and into detailed design.

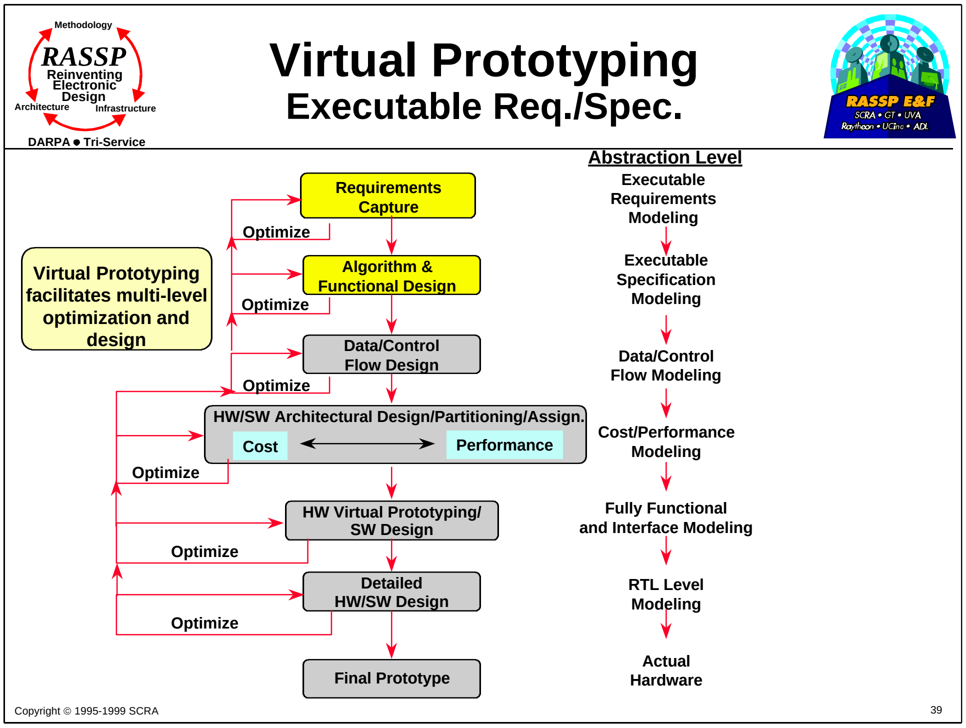
Accurate modeling of the system being developed is key to good selection of architecture and to rapid development of hardware. Good models of hardware speed development by reducing errors in design and by allowing simultaneous hardware/software development.

Detailed Section Outline

DARPA ● Tri-Service

- Overview
  - Current Design Practices
  - The RASSP Approach
- Virtual Prototyping
  - **Executable Requirements/Specifications**
    - Description
    - Case Study
  - Data/Control Flow Modeling
    - Description
    - Case Study
  - Cost Modeling
    - Description
    - Case Study
  - Performance Modeling
    - Description
    - Case Study

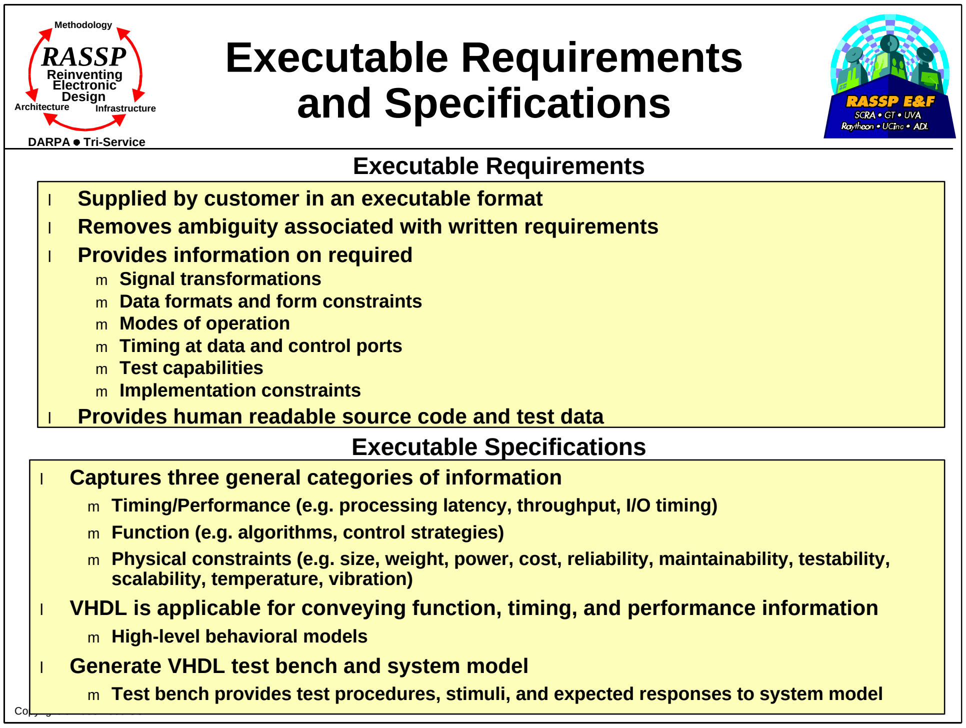
The use of executable specifications is essential to the RASSP Model Year concept which seeks to ensure that a signal processor will employ state-of-the-art technology when fielded and that it will be possible to upgrade the system throughout its lifetime. It is also a key to achieving improved design time and quality because it can provide a thread of evolving models from system definition to implementation. What constitutes an executable specification and how to name the different varieties is a subject of active discussion, but common to all definitions is simulation of the processor in its environment. A design process can be thought of as a successive refinement and adding of detail to a processor model beginning with initial requirements and ending with a virtual prototype which models the hardware and software system in complete detail. Important advantages of interoperability and reuse accrue to use of one modeling language from requirement to virtual prototype but the needs at different levels are quite different. In the requirement the algorithm may not be specified in full detail. In conceptual models there is a high premium on fast execution time to improve designer productivity. And the virtual prototype must model hardware in detail and be capable of executing application code if the device is programmable. Languages and software environments such as Matlab, Processing Graph Methodology, C, Ptolemy and VHDL are all candidates for executable specification languages. The optimum strategy for design with executable specifications has been an important focus in the RASSP community.



The functional definition phase produces a data flow model that defines the systems behavior as a set of interconnected sub-functions prior to hardware/software partitioning. These sub-function models are either used directly or translated for reuse in lower level definition phases. RASSP program has used VHDL modeling in the functional definition phase, particularly in the context of the design of a SAR image processor for purposes of benchmarking the RASSP Process. The use of VHDL modeling for a functional specification of a signal processing algorithm is unusual. It has the following advantages:

Consistent Testing Environment: Our design process is based on VHDL modeling so the functional definition is captured in the same form that the hardware development will be captured in. This allows for later side by side comparison between the functional representation and the detailed hardware design within the same environment.

Path to Synthesis: For those portions of the functional specification which will be implemented in custom hardware, rather than in a programmable processor, the VHDL description provides a better starting point for the hardware synthesis problem.



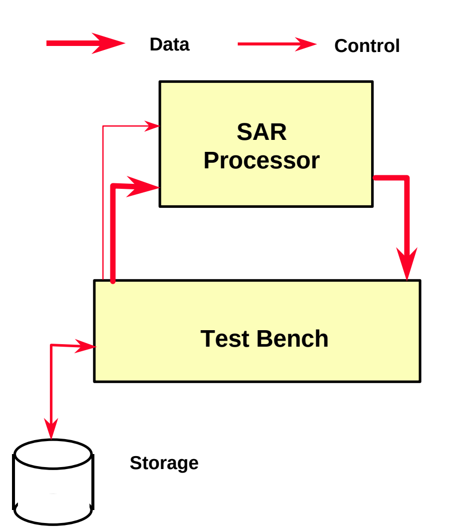
As part of the RASSP program efforts, a SAR strip map algorithm was implemented in VHDL through a straight forward translation of an existing C program. It primarily uses real and integer variables and VHDL signal variables very sparingly and executes the algorithm in zero simulated time. The VHDL created strip maps are essentially identical to those created with the C program. Data timing is modeled at the processor data input and output ports and the user can set processor latency between 0.1 and 3 seconds.

The VHDL testbench simulates the sensor system output by reformatting data from disk files and presenting it to the processor at the proper simulated time. It presents commands and setup data to the processor as a simulated host and writes output data from the processor to files and compares it with other disk file data. Latency is measured and compared with a user supplied reference. The processor and testbench model comprise 2430 lines of VHDL and use an existing math library. The VHDL processor simulation is about 25 times slower than a C program for the math parts of the SAR algorithm, that is, an FIR filter, FFTs and vector multipliers. However, not all aspects of the internal functionality will be simulated at early levels of the virtual prototyping process and the focus is on the input/output requirements/specifications and their test implications.

It is important to note the distinction between the requirements and specifications as described in the slide.

Benchmark SAR Executable Requirement

- Developed code using fully IEEE Std 1076-1987 compliant VHDL code
- Composed of the SAR processor and its corresponding test bench
- Required an estimated 1085 Mop/sec processing load
- Developed for use on Advanced Detection Technology System (ADTS) from MIT Lincoln Laboratory



The RASSP program uses benchmarks as a method to test the newly developed RASSP processes and compares them with current practice (1993). The first benchmark was a SAR processor and consisted of a test bench and processor model. The code for the models were written in fully IEEE Std 1076-1987 compliant VHDL code. The SAR algorithm required on the order of a gigaflop of computational power and was developed by MIT Lincoln laboratories.

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

Goals of Processor

- Accept data in the format of the ADTS sensor system
- Create output data in the specified format
- Model timing at the data input and output ports
- Model processor latency
- Model control modes
- Perform the processing algorithm with at least the accuracy specified in the system requirement

 42

This slide presents the goals of the processor model simulation. It accepts data in the format of the ADTS sensor, creates output data in the required format of the displays, models input and output timing as required by the specification, simulates the amount of processor latency expected by the system, models all control modes of operation, and performs the processing algorithm with at least the accuracy specified in the system requirement.

Goals of the Test Bench

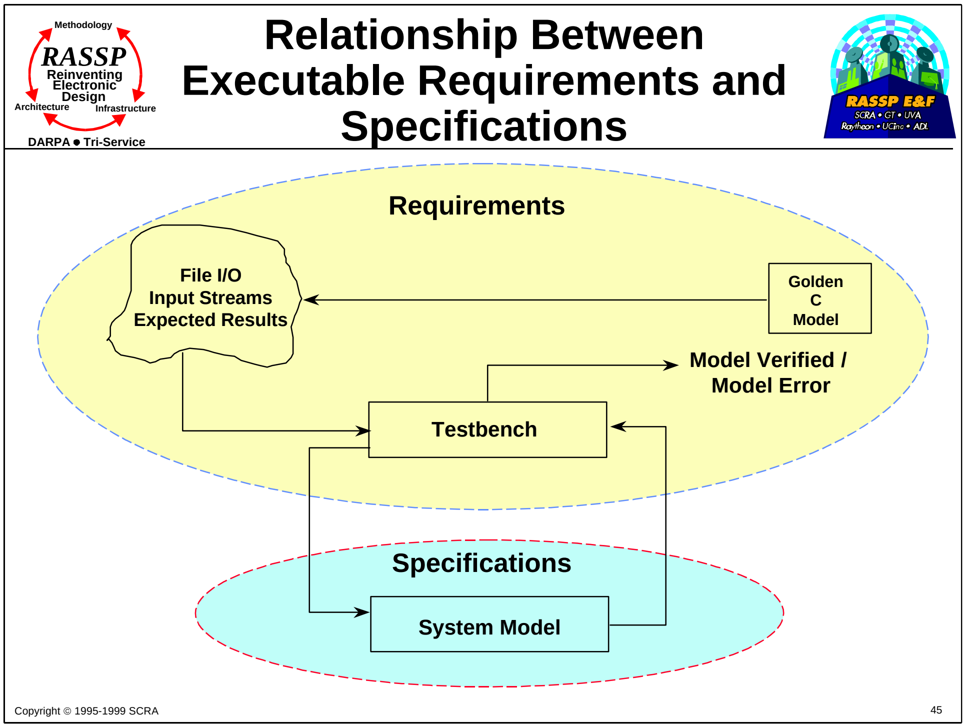
- **Control the processor from disk files with commands and setup data**
- **Read input files from disk and transform the compressed files to the ADTS format**
- **Measure processor latency**
- **Do a pixel magnitude comparison between processor output and comparison data in disk files**
- **Write output data to disk files**

The test bench controls the processor using commands and setup data read from disk files. Sensor data is modeled using disk input files and the data read from the files is transformed into that required by the ADTS system. The test bench also monitors responses of the system under test. The processor latency and pixel transformations are computed and written to disk files and compared with expected results based on the specification.

Elements in Executable Specification of the System Model

System Timing and Performance Data	System Functionality Data	Physical Constraint Data
• **Signal processing I/O data** – I/O timing constraints – I/O interface structures – I/O protocols – Signal levels – Message types • **Signal processing latency** – Data acceptance rate • **Signal processing stimuli/response**	• **Algorithm descriptions** • **Control strategies** • **Task execution order** • **Synchronization primitives** • **Inter-process communication (IPC)** • **BIT and fault diagnosis**	• **Size** • **Weight** • **Power** • **Cost** • **Reliability** • **Maintainability** • **Testability (fault coverage, diagnosis, and BIST goals)** • **Repairability** • **Scalability** • **Environment constraints** – Temperature – Vibration – Pressure – Stress and Strain – Humidity – EMI/EMF/EMP

The above slide represents the organization of the information in an executable specification of a system. We suggest that the above information be included as part of the executable specifications capture process.

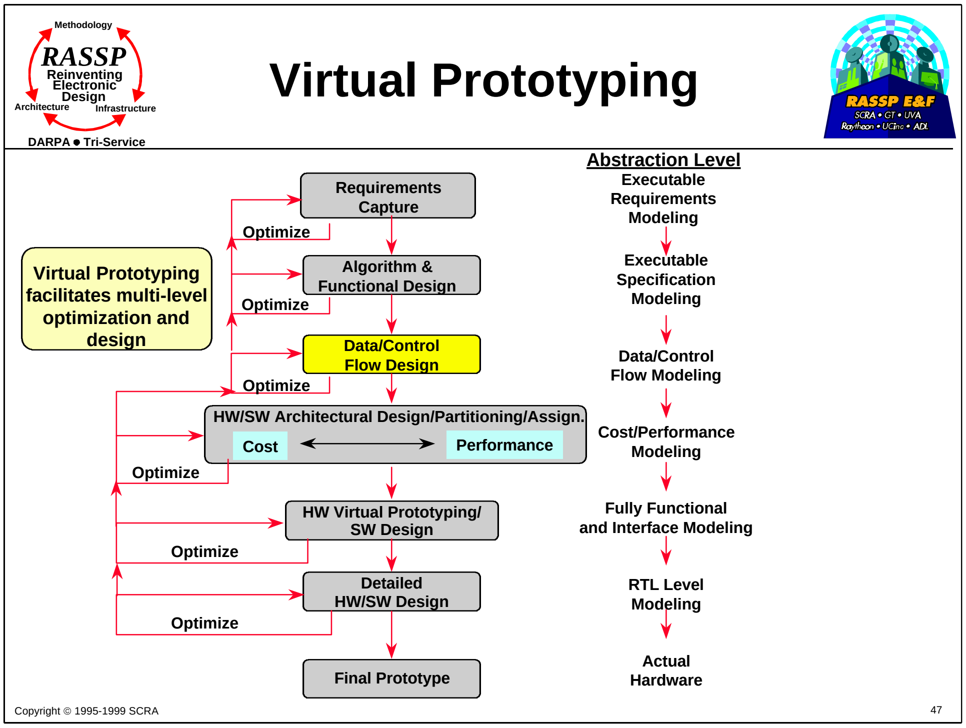


The relation between the executable specifications and requirements is shown above. The requirements describe what a system should do, while the specifications describe the model of the system (including details of its implementation) that require that it satisfy the same test bench utilized as part of the executable requirements capture process.

Detailed Section Outline

- **Overview**
  - **Current Design Practices**
  - **The RASSP Approach**
- **Virtual Prototyping**
  - **Executable Requirements/Specifications**
    - **Description**
    - **Case Study**
  - **Data/Control Flow Modeling**
    - **Description**
    - **Case Study**
  - **Cost Modeling**
    - **Description**
    - **Case Study**
  - **Performance Modeling**
    - **Description**
    - **Case Study**

The Data/Control Flow phase follows the Executable/Specifications phases.

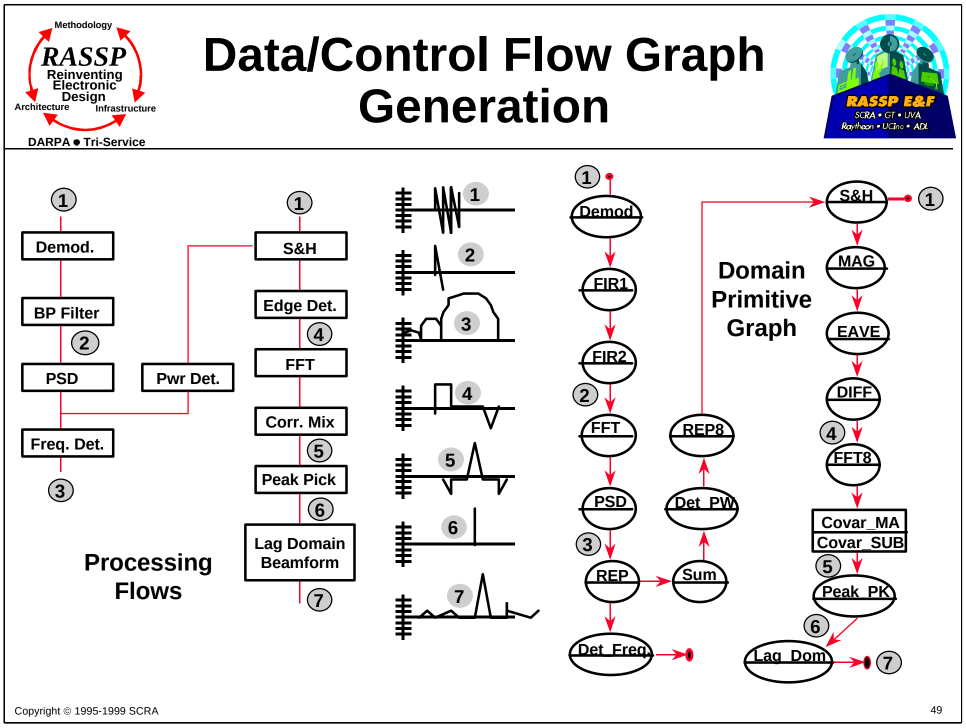


During Data/Control flow graph (DFCG) modeling the internals of the executable specification are represented in an executable form to highlight features such as concurrency. A DFCG describes an application algorithm in terms of its inherent data dependencies of its mathematical operations. The DFG is a directed graph containing nodes that represent mathematical transformations and arcs that span between nodes and represent their data dependencies and queues. It conveys the potential concurrencies within an algorithm, which facilitates parallelization and mapping to arbitrary architectures. The DFG is an architecture independent description of the algorithm. It does not presume or preclude potential concurrency or parallelization strategies. The DFG can be a formal notation that supports analytical methods for decomposition, aggregation, analysis and transformation.

Data/Control Flow Modeling

- **Contained in the architecture selection design process**
- **Accepts the algorithms processing flows as input from the system design process**
- **Generates implementation-independent representation of the system data flow**
- **Generates information for data flow graph control**
- **Verify virtual prototype at this level using data passed down from the previous design phase**

The DFG nodes usually correspond to DSP primitives such as FFT, vector multiply, convolve or correlate. The DFG graph can be executed by itself in a data-value-true mode without being mapped to a specific architecture, though it can not resolve temporal details without co-simulation with an architecture performance model. The primary purposes of a data flow graph are to express algorithms in a form that allows convenient parallelization and to study and select optimal parallelization or execution strategies through various methods involving the aggregation, decomposition, mapping and scheduling of tasks onto processor elements and data flow aspects of the behavior inside the functional virtual prototype.



An example of a Data Flow Control Graph representation of a signal processor is shown above using the Processor Graph Methodology (PGM) proposed by the US Naval Research Laboratory in the late 70s and early 80s. The functional process flow is described on the left and its implementation using domain primitive graphs is described on the right. The specification on the right is expected to be in a executable format.

Flow Graph Generation and Simulation Mechanism

- **PGM Tools: GRED and GRAIL**
  - **Processing Graph Method (PGM)**
  - **GRED is a graphical editor for building PGM graphs**
  - **GRAIL is a translator from graphical format to Signal Processing Graph Notation (SPGN)**
- **PGSE: Processing Graph Simulation Environment**
  - **Functional simulation of PGM graphs**
  - **Provides standard interface to command program**
  - **Provides ability to simulate command program interacting with multiple PGM graphs**

The PGM environment provides for the GRED and GRAIL utilities that assist in the composition of the executable specification from domain primitives. The PGSE is the simulation environment that links the functional specification to the control/sequencing that complete the executable specification.

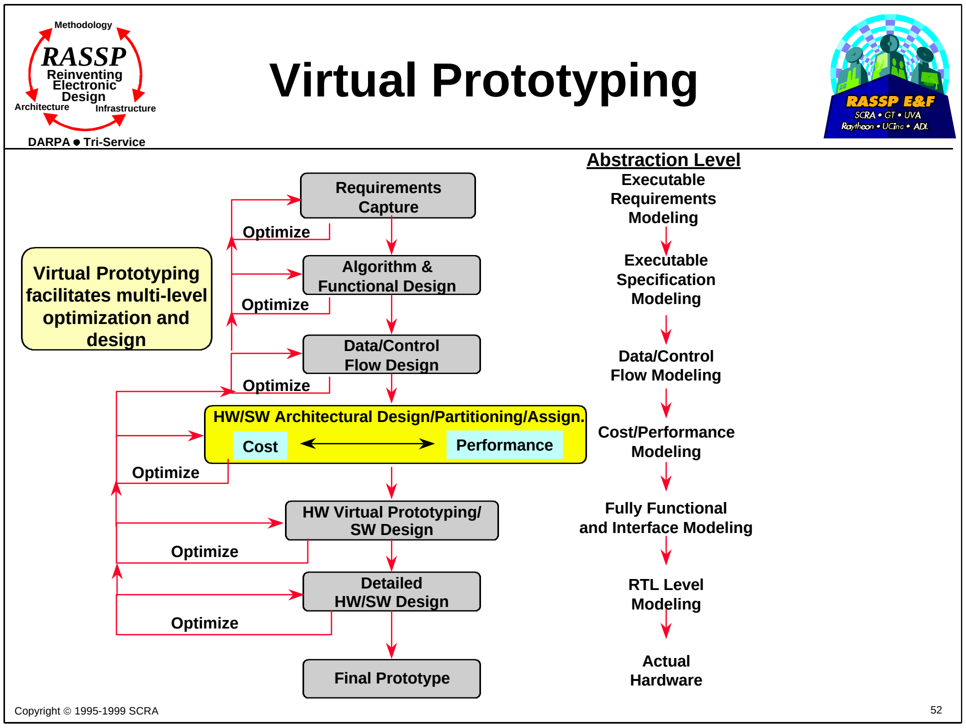
RASSP Reinventing Electronic Design — Methodology, Architecture, Infrastructure — DARPA ● Tri-Service

Detailed Section Outline

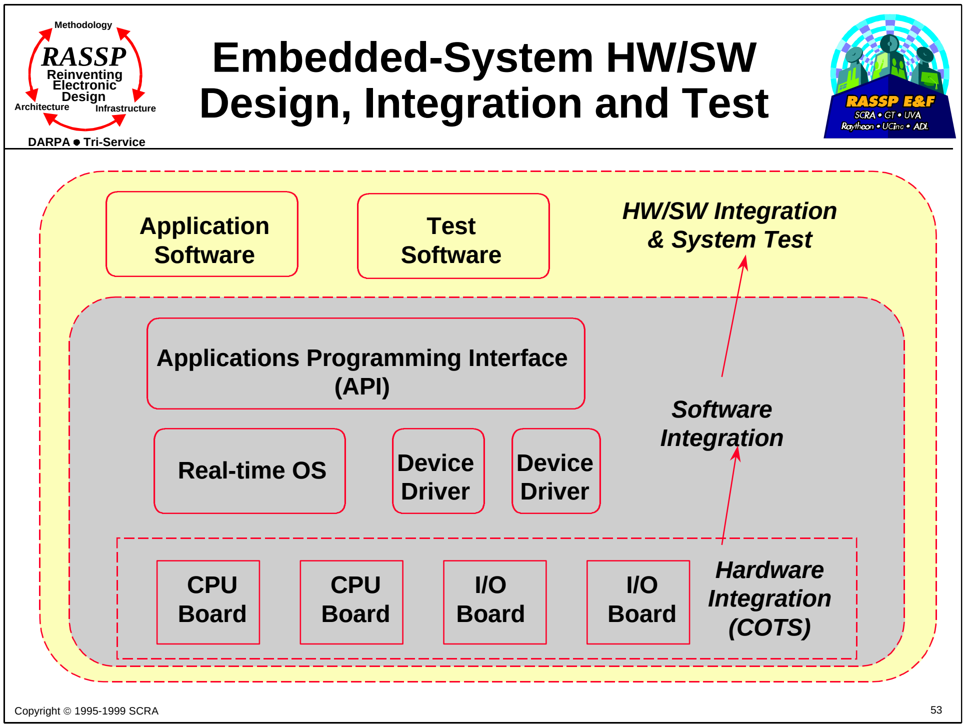
- Overview
  - Current Design Practices
  - The RASSP Approach
- Virtual Prototyping
  - Executable Requirements/Specifications
    - Description
    - Case Study
  - Data/Control Flow Modeling
    - Description
    - Case Study
  - **Cost Modeling**
    - Description
    - Case Study
  - Performance Modeling
    - Description
    - Case Study

 51

Cost Modeling evaluates the cost (in terms of design time, design costs, life-cycle costs and HW/SW costs, to name a few) of various possible architectural candidates for implementation. Cost can be an independent variable in this decision making process, and involves modeling various aspects of the design and lifecycle costs in a form that allow them to be included in the virtual prototyping process early on the in the system design process.



We will focus on the cost modeling phase of the virtual prototyping process, wherein cost modeling is used to synthesize candidate architectures based on architectural partitioning, allocation, and scheduling algorithms, followed by performance verification through performance modeling and simulation. For a detailed discussion on Cost Modeling see J. DeBardelaben, V. Madisetti, and A. Gadient, "On Incorporating Cost Modeling in Embedded Systems Design," *IEEE Design & Test of Computers*, Vol. 13, No. 3, July 1997.



The VP process spans multiple levels and multiple user's viewpoints.

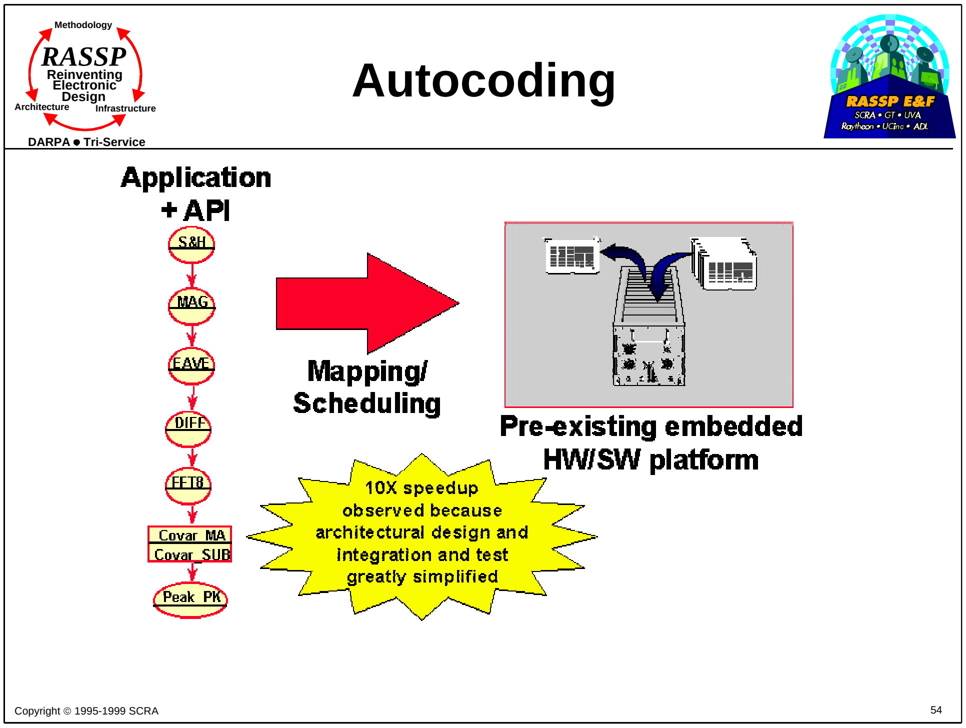
At the lowest level of HW integration, we have HW design being done and one would typically see the following being used:

- Full-behavioral/Interface and RTL level models of application specific and COTS parts being modeled
- Interconnection between devices are tested

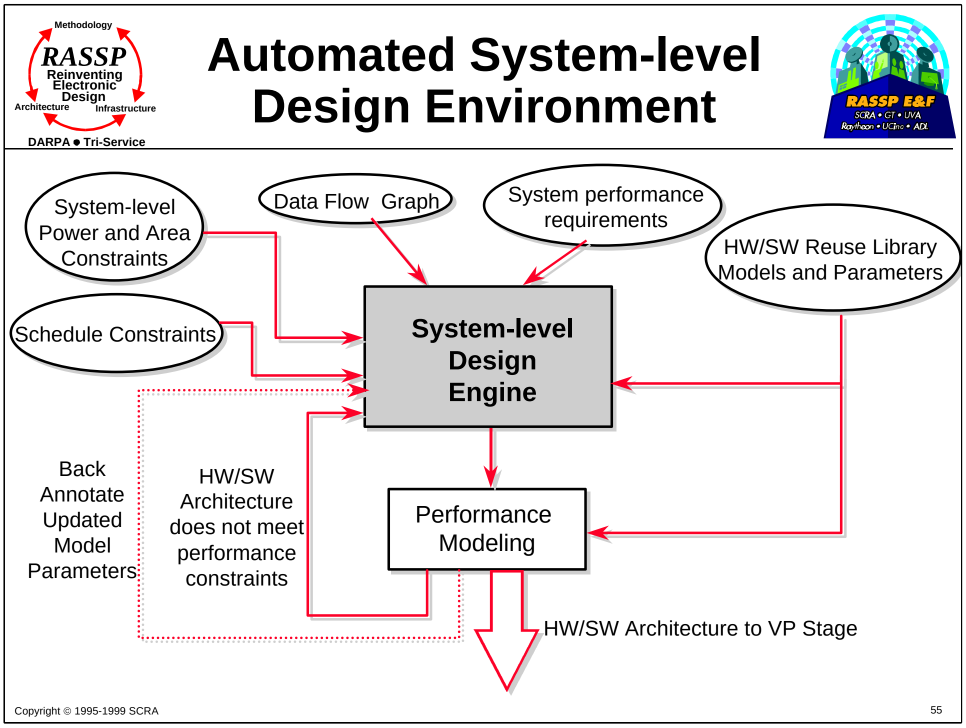
The next level integrates the OS SW and application interface to the HW system. This is where one would see SW running on the HW VPs to make sure the device drivers work as expected.

At the highest level, application and test code is integrated and tested at the system level. Performance level models help determine the number of processing boards required. Full-behavioral models help insure test SW can perform its functions at the node level. Executable specification helps determine the application SW.

The VP process covers all these domains (See Madisetti & Egolf, IEEE Micro, Fall 1995).

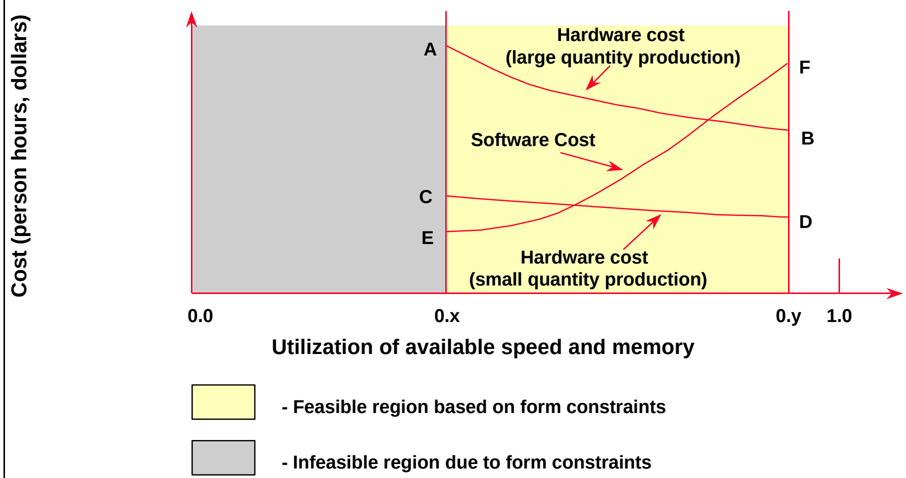


In case the architecture of the target platform is not fixed, then the architecture has to be synthesized and verified prior to code generation and mapping. This requires cost modeling to synthesize the candidate architectures, and performance modeling to verify each candidate architecture. Extensive use of re-use libraries for cost, performance and functionality is done in this phase of the virtual prototyping process.



In case the architecture of the target platform is not fixed, then the architecture has to be synthesized and verified prior to code generation and mapping. This requires cost modeling to synthesize the candidate architectures, and performance modeling to verify each candidate architecture. Extensive use of re-use libraries for cost, performance and functionality is done in this phase of the virtual prototyping process.

HW/SW System Prototyping Costs



 56

• The graph shows that hardware constrained architectures can significantly increase total system costs especially in systems which are produced in small quantities, such as systems for military applications.

• Reasons for increased software cost:

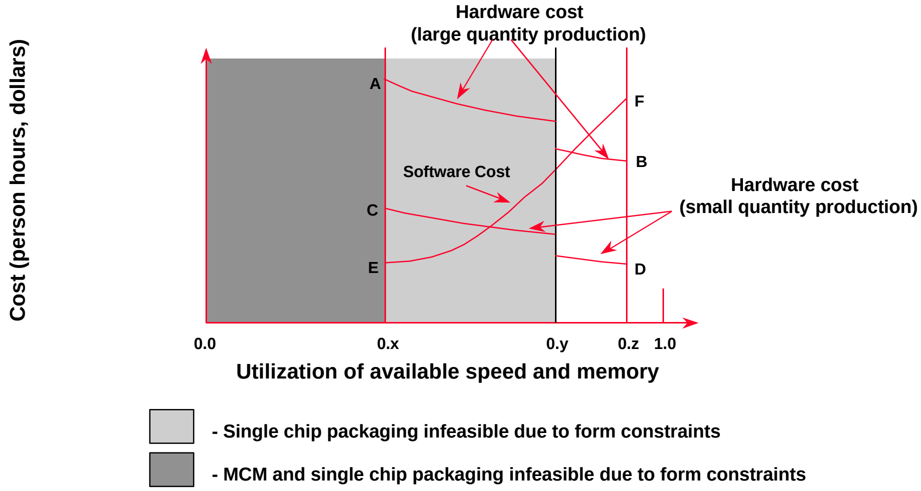
- code and data is trickier to program and debug
- more complex test procedures, harder test drivers and diagnostics
- added analysis, simulation, prototyping, validation
- added performance measurement functions
- complex resource management
- tight execution time budget and memory core control

• If physical constraints permit, the hardware platform can be relaxed to achieve significant reductions in overall development cost and time.

• Many parametric software cost models support this principle of software prototyping.

Further details are available in J. DeBardelaben, Incorporating Cost Modeling in Embedded Systems Design," *IEEE Design & Test of Computers*, Vol 13, No. 3, July 1997.

The Effect of Packaging on Prototyping Costs



 57

- Multi-chip module technology allows for increased packaging density over single-chip packaging.
- This increased packaging density can allow for more slack to be added to the hardware architecture without violating system-level form factor constraints.
- This added slack margin can possibly lead to significant software cost reductions.
- However, the reduction in software cost is traded off against the increase in production costs due to MCM manufacturing.

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

REVIC Software Development Cost/Schedule Model

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

REVIC Embedded Mode Model

Software Development Cost

$$S_c = L_c\left[4.44(KDSI)^{1.2} F_E \bullet F_M \bullet F_{MODP} \bullet F_{TOOL} \prod_{i=5}^{18} F_i\right]$$

Software Development Time

$$S_T = 6.2\left[4.44(KDSI)^{1.2} F_E \bullet F_M \bullet F_{MODP} \bullet F_{TOOL} \prod_{i=5}^{18} F_i\right]^{0.32}$$

Execution Time and Main Storage Constraint Effort Multipliers

Rating	Utilization	F_E	F_M
nominal	up to 50%	1.00	1.00
high	70%	1.11	1.06
very high	85%	1.30	1.21
extra high	95%	1.66	1.56

- The parametric equations used by REVIC software cost model quantitatively describe the relationship between software cost and hardware utilization.
- As the hardware utilization increases, so does the value of the execution time and main storage constraint multipliers. This increase causes a corresponding increase in the software development cost and time.
- Also, the model describes an associated increase in HW/SW integration cost and time.
- In the above REVIC model, the development cycle includes the contract award through hardware/software integration and testing.
- The software development time equals the system development when hardware platform consists of mostly COTS hardware components.
- The units of development cost and time are person-months and calendar months, respectively.

REVIC Software Development Cost/Schedule Model

Factors Strongly Influenced by RASSP Tools and Methodology

Modern Programming Practices

Rating	Description	F_{MODP}
very low	no use	1.24
low	beginning or experimental use	1.10
nominal	experienced in use of some	1.00
high	experienced in use of most	0.91
very high	routine use of all modern programming practices	0.82

Software Tool Usage

Rating	Description	F_{TOOL}
very low	very few - primitive tools	1.24
low	basic microcomputer tools	1.10
nominal	basic minicomputer tools	1.00
high	Basic maxicomputer tools	0.91
very high	extensive tools but little integration	0.83
extra high	moderate tools with integrated env.	0.73
extremely high	fully integrated environment	0.62

 59

- The RASSP methodology strongly supports the use of modern programming practices and advanced software tools. Modern programming practices include reuse-driven development methodologies, spiral development, incremental development and/or object-oriented approaches. RASSP supports a fully integrated software tool suite including:
  - VHDL performance models, instruction set architecture models, executable specifications.
  - Life cycle tools which are fully integrated with processes, methods, and reuse.
- However, tight hardware resource constraints can have adverse effects on the use of advanced software development tools. The use of high order languages and compilers is very restricted when execution time and memory margins decrease. Especially, the inefficiency of many DSP compilers prohibits their use when memory core and execution time budgets are very tight.

Design Example: SAR Processor Requirements

- **Software Size**
  - **8,750 lines of uncommented source code**
- **Performance requirement**
  - **1 billion floating point operations per second**
- **Size limit**
  - **10.5" height x 20.5" length x 17.5" width**
    - **Sufficient to hold 21-slot 6U VME card cage, etc.**
- **Average power limit**
  - **500 watts**
- **Weight limit**
  - **60 pounds**

The SAR processor example has been used as part of the RASSP Benchmark efforts to measure metrics on the efficacy of the virtual prototyping process. The input requirements for the SAR processing efforts are provided above in terms of form, fit and function, together with some performance and code size values.

Dependence of Objection Functions on Customer Class

DARPA ● Tri-Service

- **Commercial Applications**
  - **<u>Objective</u> - select an architecture which will maximize profits subject to design constraints**
    - **Profit = Total potential revenue - HW/SW DEV/PROD cost**
- **Government Applications**
  - **<u>Objective</u> - select an architecture which will minimize life cycle costs (LCC) subject to design and schedule constraints**
    - **LCC = HW/SW Dev/Prod. Cost + HW/SW Maintenance Cost**

This slide shows the differences in the objective functions of commercial and government/military type applications. The virtual prototyping, as espoused by the RASSP process, is applicable to both applications, with the provision that different objective functions be included in the early architectural synthesis process. The virtual prototyping ideas are also equally applicable to the System-On-Chip (SOC) and Systems-on-Chip (SOP) design efforts.

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

SAR Multiprocessor

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **SAR Processor is required to form images in real-time on board an F-22 or an unmanned air vehicle (UAV)**
- **Pulse Repetition Frequency - 556 Hz**
  - **Delivers 512 pulses in 0.92 seconds**
  - **Each pulse contains 4064 data samples for each of four polarizations**
- **Processor must be able to form a 512-pulse image for each of three polarizations in real-time**
  - **Maximum latency is 3 seconds**
- **Computational Requirement - 1.1 Gflop/sec**
- **Memory Requirement - 77 MB**

 62

This slide provides some specific details of the executable requirements for the SAR benchmark application.

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

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Manual Tradeoffs

- Mercury MCV6 cards
  - Four 40 MHz Intel i860 processors per card
  - Each card weighs 1.875 pounds
  - Maximum power dissipation of 28W per card
    - Up to 14 cards could be used in the system
  - Variable memory configurations

Model	Memory (Mbytes DRAM)	Price ($)
4x4m	16	30,200
4x8m	32	35,800
4x16m	64	47,000

 63

Several target architectures are possible for the SAR application. One can design a custom hardware architecture, and then develop software for that architecture. One can also use off-the-shelf (COTS) signal processing boards (such as Mercury MVC6) cards that come with supporting control and diagnostic environments that assist in code development. Since the virtual prototyping approach facilitates the use of cost as an independent variable, we also enumerate the costs in terms of hardware costs and the design costs for software developed during the prototyping phases.

At least three objective functions may be formulated:

1. A minimum HW cost system

2. A reduced cost system (that improves upon the minimum cost system, while relaxing a few constraints that contribute to the total system costs).

3. A minimum HW+SW cost system(possibly the desired objective).

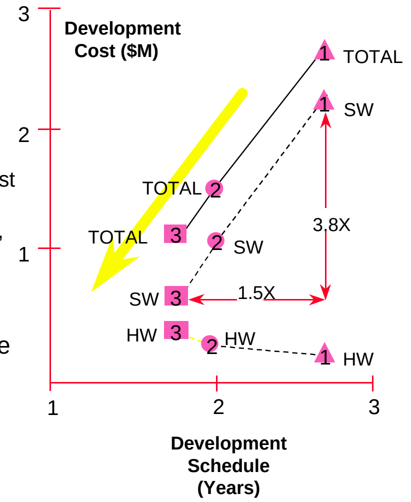
RASSP Reinventing Electronic Design — DARPA • Tri-Service

Design Tradeoff Case Study: SAR Processor with COTS Multiprocessor Cards

1 Minimum Hardware Cost
- Lowest Production Cost
- Minimum size, weight, and power
- Nominal memory and computational margins

2 Reduced Development Cost & Time
- Same number of cards and life cycle cost as 1
- RAM added to improve memory margin, allow use of advanced software development tools and methodology

3 Minimum Development Cost and Time
- Processors added to 2
- Improved computational margin further eases software development



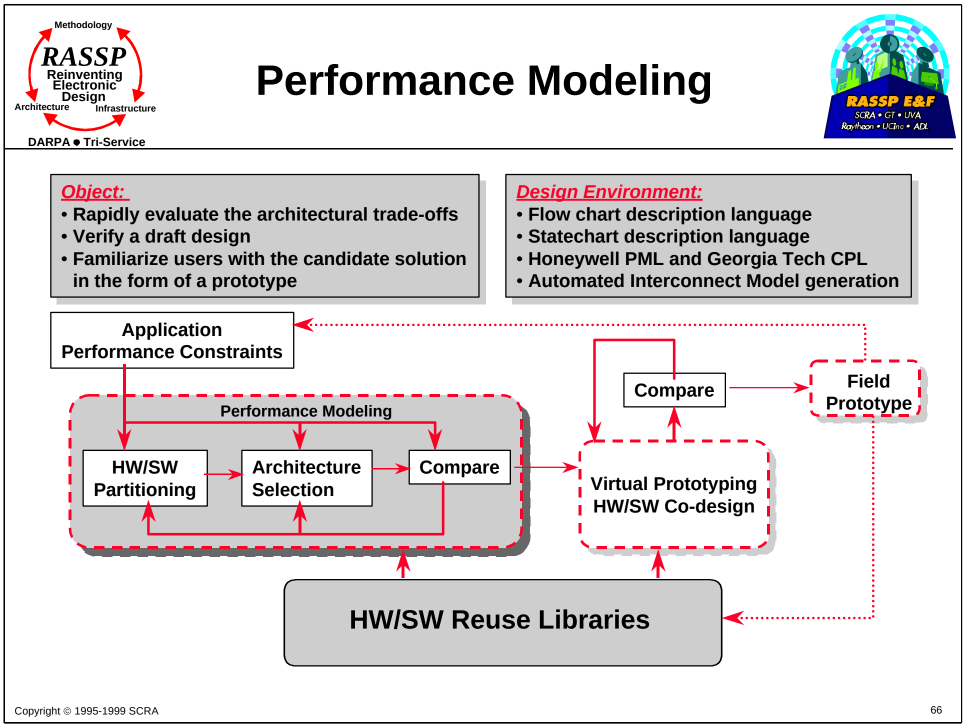
Copyright © 1995-1999 SCRA 64

- In all three cases, use of a commercial off-the-shelf multiprocessor card solution is assumed, but details of the processor and memory margins differ. If the designer focuses entirely on minimizing hardware cost, the software development cost is nearly four times that of a design which seeks to minimize the overall development cost and time. The curves compare development cost and time for a synthetic aperture radar processor under three different assumptions regarding computation and storage requirements. The minimum hardware cost implementation uses only six MCV6-4x4m cards with a 88% execution time utilization and 86% memory utilization. Resulting hardware component cost is $100,000 plus the cost of the six cards - $281,000. Software cost development cost and time are $2,360,000 an 32 months, respectively (total cost - $2,640,000). The reduced development cost/time implementation uses six MCV6-4x8m cards, thereby decreasing memory utilization to 43% allowing for the use of advanced software development tools and methodology. Software development cost and time decrease to $1,030,000 and 24 months; while hardware cost slightly increases to $315,000 (total cost - $1,350,000). The minimum development cost/time implementation uses eleven MCV6-4x4m cards with less than 50% memory and processor utilization. This further reduces software development cost and time to $620,00 and 21 months; while hardware increases to $432,000 (total cost - $1,050,000).

Detailed Section Outline

- **Overview**
  - **Current Design Practices**
  - **The RASSP Approach**
- **Virtual Prototyping**
  - **Executable Requirements/Specifications**
    - **Description**
    - **Case Study**
  - **Data/Control Flow Modeling**
    - **Description**
    - **Case Study**
  - **Cost Modeling**
    - **Description**
    - **Case Study**
  - **Performance Modeling**
    - **Description**
    - **Case Study**

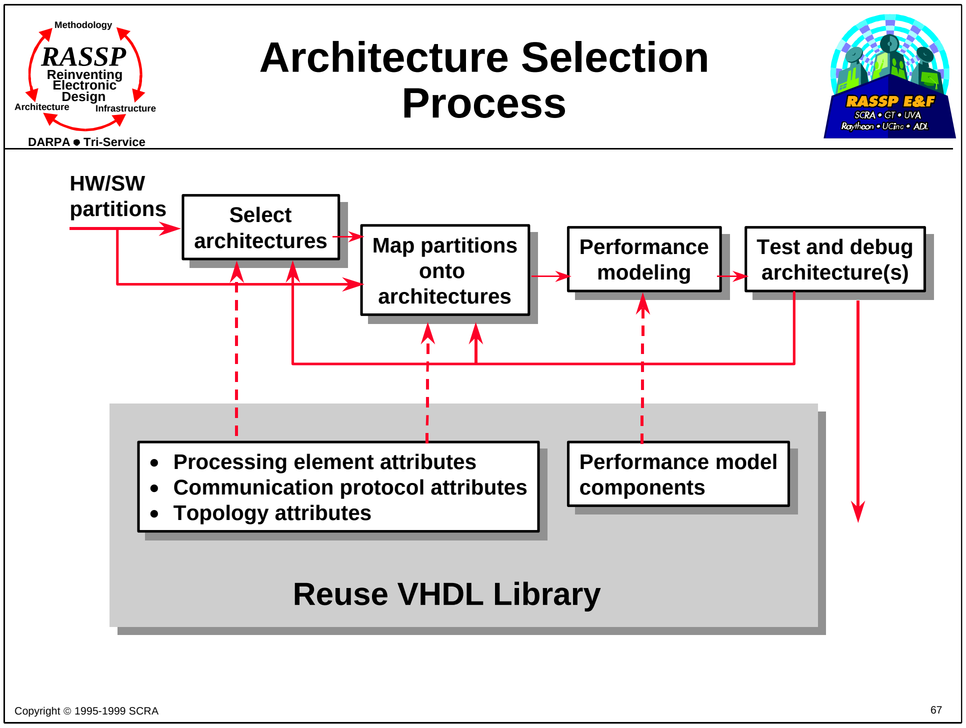
Once cost modeling generates candidate architectures for HW and SW, performance modeling plays an important role in verifying that these architectures meet with performance constraints (e.g., throughput, sample rate, etc.).



The intention of performance modeling is to explore and verify architectural tradeoffs such that the application's performance needs are met with efficiency. Using the conceptual prototyping technology based on performance modeling, system developers are able to determine the proper architecture components and predict system performance before developing the embedded systems. Thus, conceptual prototyping does improve the cost and design times of embedded systems dramatically.

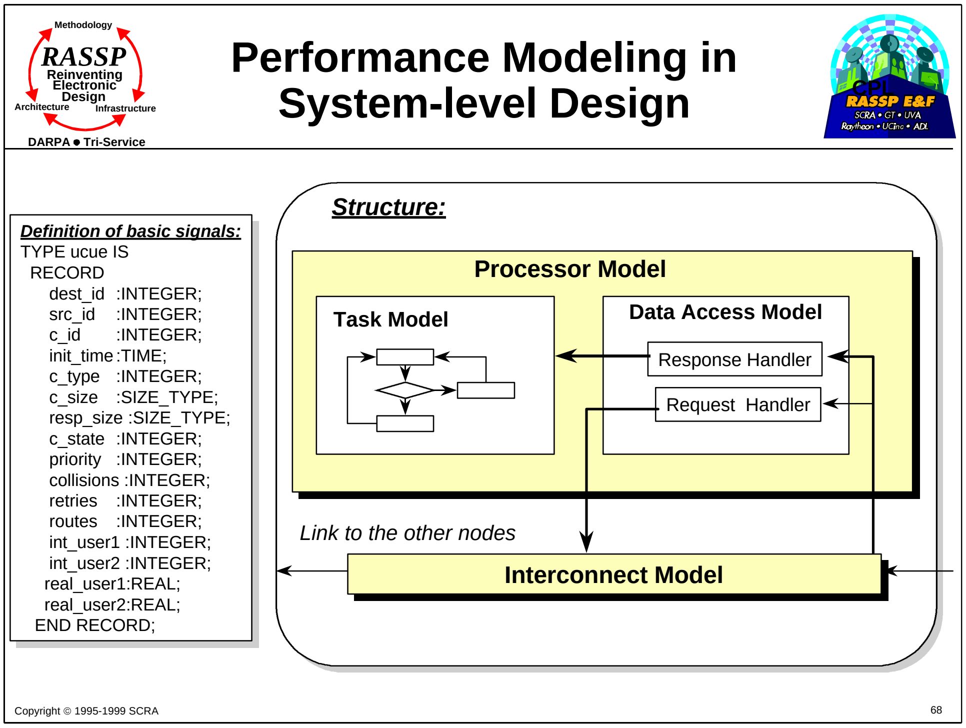
RASSP's utilizes a VHDL-based executable performance-level models as part of an interoperable validated RASSP component library.These performance models do not really transfer data streams or access handshaking signals, but do detect the states of components and transfer virtual packets between components. Each virtual packet contains the information of a transaction, such as the source , destination , packet size, packet identification number and packet status.

Via simulation and debugging the performance-level models, conceptual prototypes are generated by comparing candidate architectures in terms of performance metrics, such as throughput, latency, utilization and scalability.



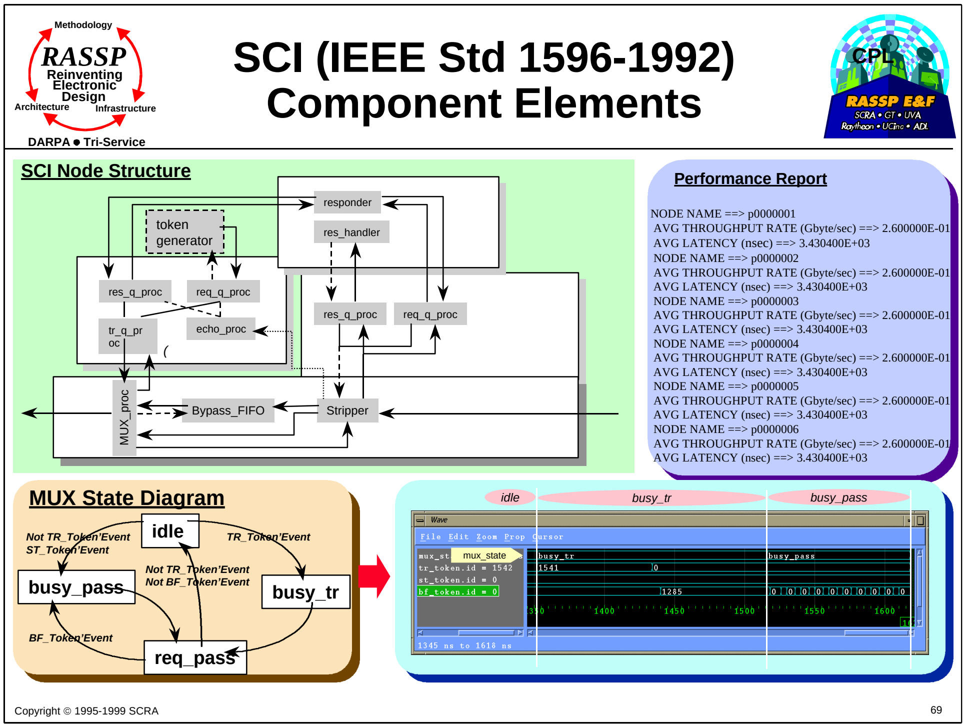
After selecting some candidate architectures and verifying their conceptual prototypes, system developers can acquire the performance results and select the most preferred architecture.

For further details see L-R. Dung, V. Madisetti, "Conceptual Prototyping of Scalable Embedded DSP Systems," IEEE Design & Test of Computers , Vol 13, No 3., Fall 1996.



The typical embedded platform consists of three fundamental components - processing elements, communication protocols and configuration. The performance models of processing components are created by simply setting the component attributes, such as the processor throughput and latency,specific task processing time, and memory access time, and configurations can be done by VHDL port mapping.

However, the critical part of our performance modeling is the implementation of communication protocols; we need to convert the communication protocol to some flow charts or state diagrams and then write VHDL processes to realize the flow charts or state diagram as shown in the above slide.



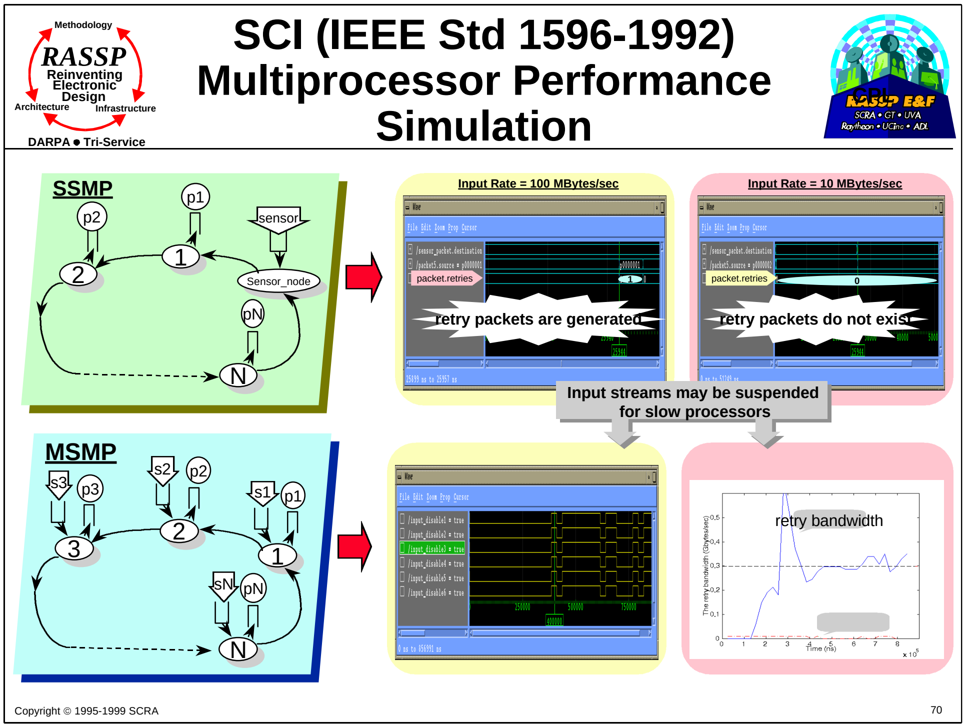
In general, there are four basic elements in communication components - requester, responder, arbiter and the handshaking signals. The handshaking signals control the data flow and transactions among components and indicate the states of components or the types of transactions.

The slide shows how these various functional blocks are captured for the SCI (IEEE Std 1596-1992) communications protocol.

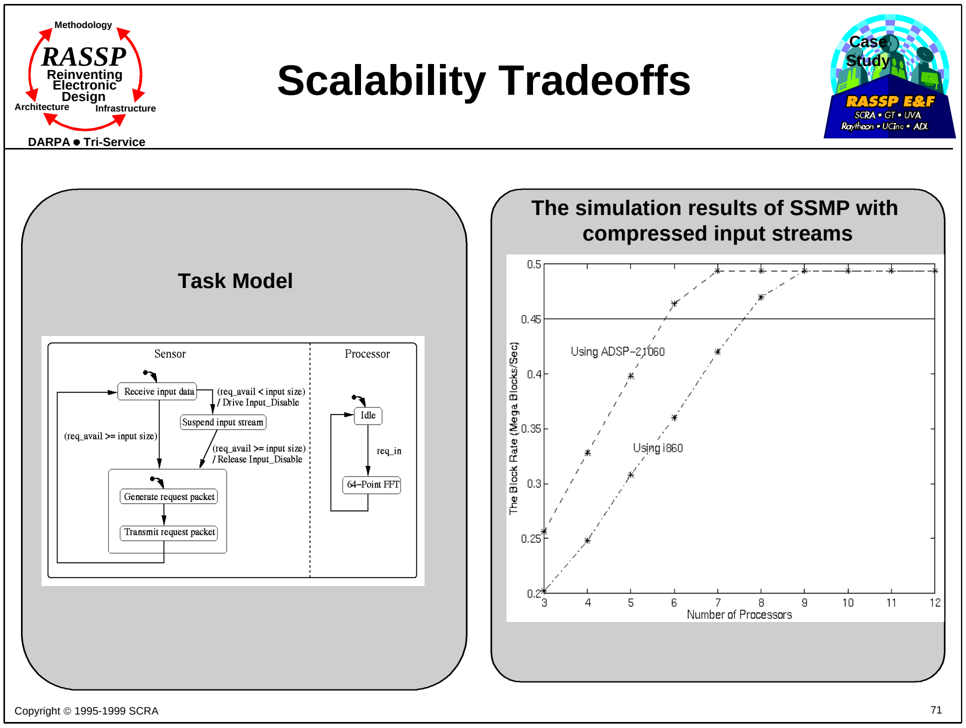
The basic SCI interconnect model is shown above in the pm_scinode. The linc is a link channel that receives packets from the preceding interconnect model and sends packets to the successive node.

A linc contains three components: stripper, bypass FIFO and a MUX. The stripper selectively strips incoming packets, creates echo packets and replaces the selected non-idle symbol by an idle symbol. The bypass FIFO is employed to delay pass-through packets while a transmit-queue packet is being sent. The MUX is used to determine the output packet among the queuing input packets.

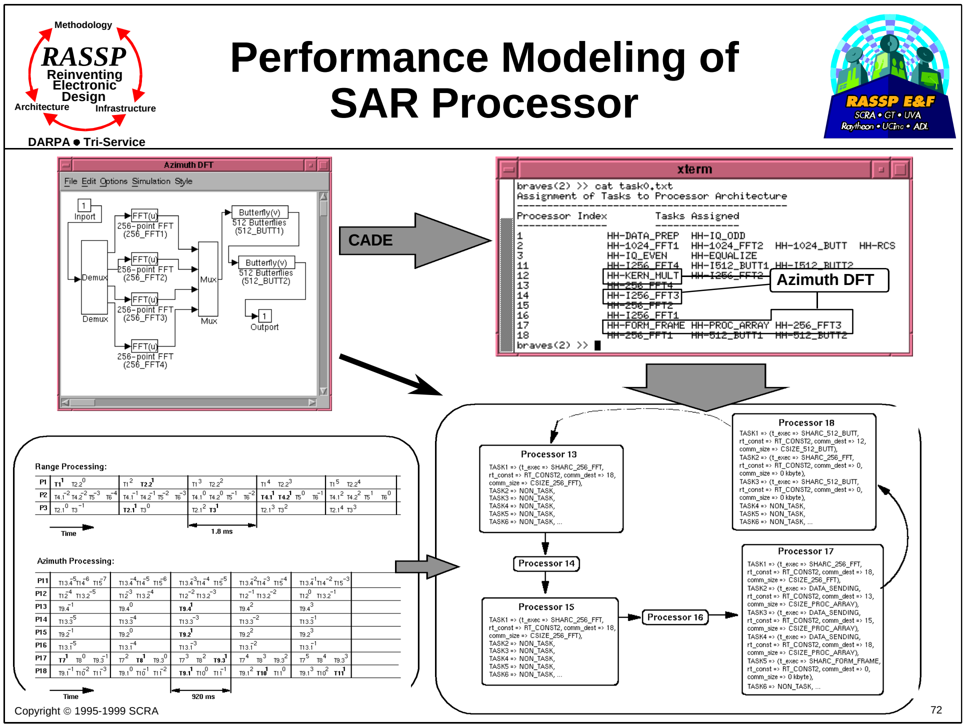
For further details see L-R. Dung, V. Madisetti, "Conceptual Prototyping of Scalable Embedded DSP Systems," *IEEE Design & Test of Computers* , Vol 13, No 3., Fall 1996.



In the above slide two architectures for a sensors-based multiprocessor platform are evaluated. The performance of a real-time scheduling protocol is evaluated and it is observed that in one case "retry" packets are generated implying that the system has errors and may not meet real-time processing constraints due to non-determinism. Plots of bandwidth analyses and processor vs. performance tradeoffs can be quickly explored using the performance modeling environment (as shown in the next slide).



Using an executable representation of SW executing on various nodes (using a high-level pseudo code) one can evaluate HW/SW partitioning tradeoffs prior to developed detailed software and hardware designs, facilitating quick early tradeoffs and scalability analyses.

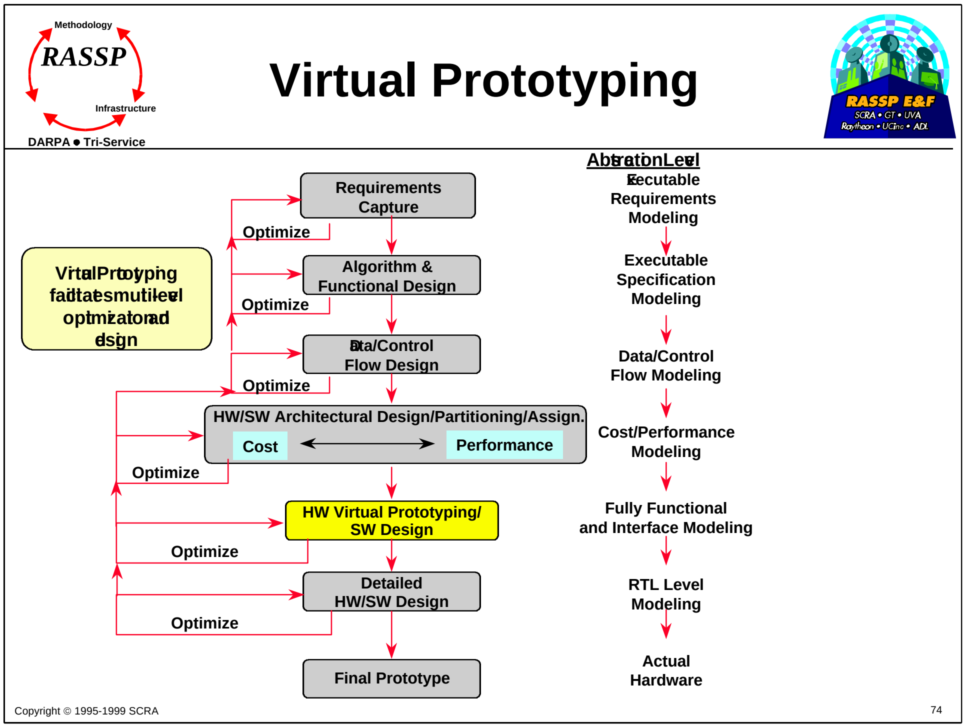


The performance modeling activity was carried out for the SAR benchmark as shown above that results in an acceptable schedule and partitioning after the performance modeling effort. This tradeoff was carried out at Georgia Tech's RASSP Laboratory at the Center for Signal and Image Processing.

Detailed Section Outline

- **Fully Functional Modeling**
  - Description
  - Case Study
- **Model Year Architecture**
- **Reuse**

We now describe the next stage of virtual prototyping (after the performance modeling/cost modeling-based HW/SW partitioning and architectural design has been completed).



The detailed behavioral (= fully functional and interface) model is a behavioral model that describes the component's interface explicitly at the pin level. It exhibits all the documented timing and functionality of the modeled component, without specifying internal implementation structure. This type of model has traditionally been called a full-functional model and is therefore a synonym. However, the newer term is preferred for its better accuracy and consistency to the definitions of the related models.

The primary purpose of a detailed behavioral model is to develop and comprehensively test the structure, timing and function of component interfaces, especially of a board level design. Also to examine the detailed interactions between hardware and software (drivers), and to provide timing values that are used to replace initial estimates in the higher level models to increase their accuracy.

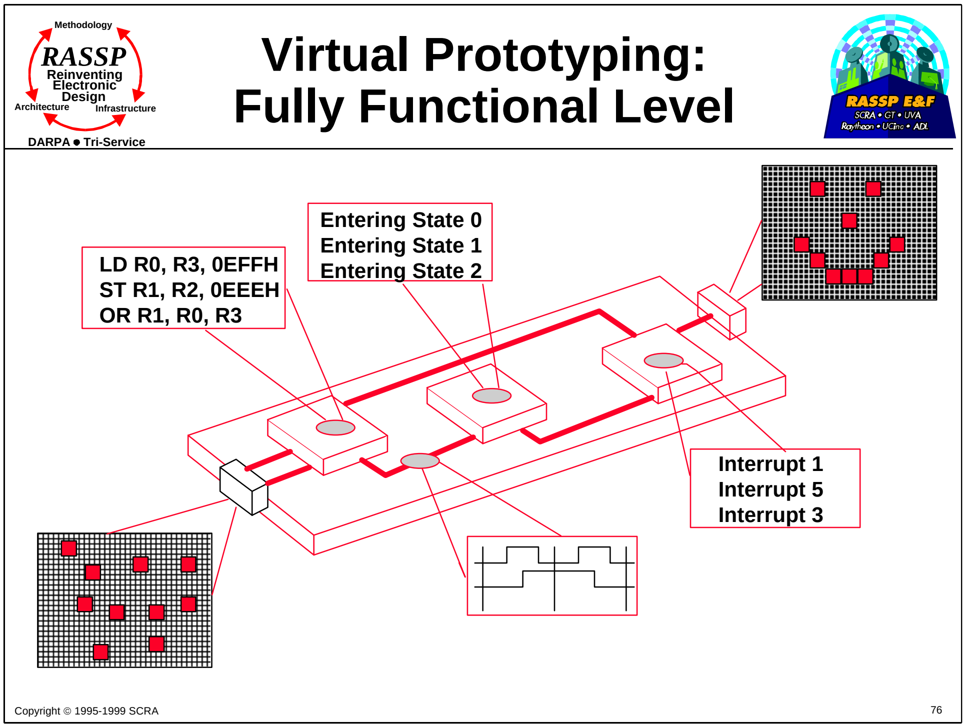
RASSP Reinventing Electronic Design
Methodology · Architecture · Infrastructure
DARPA ● Tri-Service

Virtual Prototyping at the Fully Functional and Detailed Levels

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

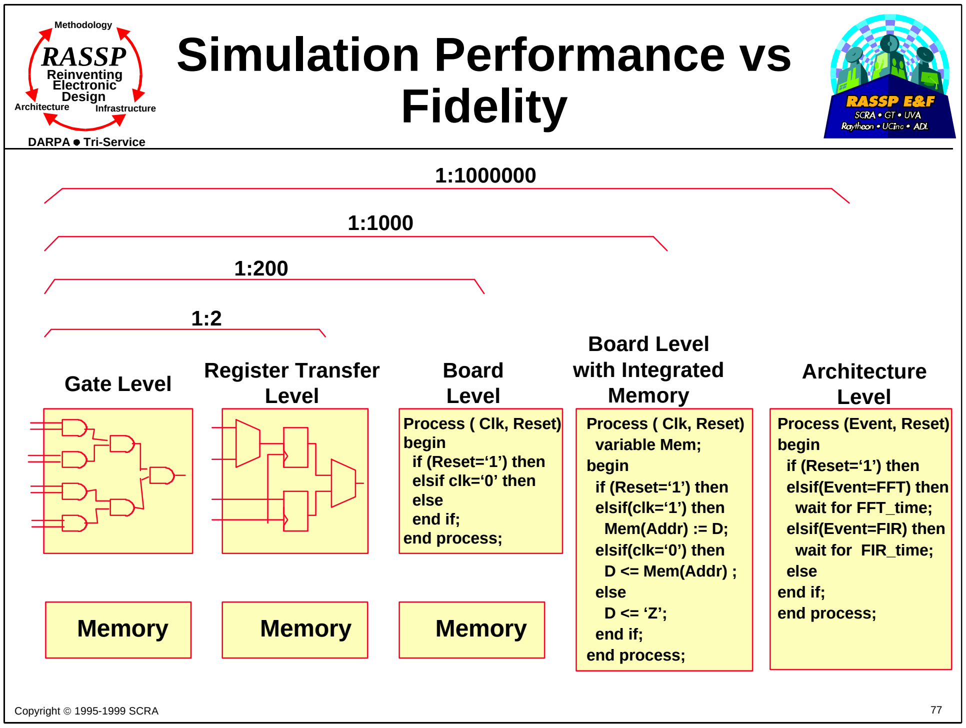
- **What ?**
  - **Software "simulatable" model of a hardware component, board, or system containing sufficient accuracy to guarantee its successful realization in hardware**
- **Why ?**
  - **Eliminate hardware prototyping from the in-cycle design loop**
  - **Allow for concurrent design of HW and SW**
  - **Provide rapid HW/SW integration**
  - **Maintain the ARPA thrust through RASSP**
  - **Document the system characteristics for future supportability**

Virtual prototyping at the fully-functional and detailed levels is used to facilitate early software development and also to ensure simpler and easier hardware/software integration and test. While several methodologies support the use of virtual prototyping, the primary impediment to the success of virtual prototyping is the availability of libraries of detailed models of hardware and software.



A Full Functional Model provides a model that is structurally correct and exhibits the functional and performance characteristics of the entities being modeled. At this level dedicated hardware elements are modeled by their behavior, not in a way that implies their implementation. For example, a dedicated filtering chip would be modeled in way that was correct bit-wise but did not imply the implementation structure.

At this level of modeling, RASSP has proposed using Instruction Set Architecture (ISA) and Instruction Set Simulator (ISS) models. The ISA modeling approach is to develop VHDL behavioral models of a processor that can execute software and provide complete access to the internal registers of the processor [5]. The ISS modeling approach is to integrate a commercial processor simulator into a VHDL environment. In either case, the processor model is combined with a Bus Interface Model(BIM), which models the detailed interaction of the processor at its connections, to make the full functional model. This approach both to model individual chips, i.e., the i860, and to model single board computers (See Madisetti & Egolf, IEEE Micro, Fall 1995).



Both the ISA and the ISS approaches allow the application software that is to run on the target hardware to be run in the simulation environment prior to physical hardware delivery. It is at this stage in the modeling process that detailed errors about the meaning of interfaces can be identified and corrected. Additionally, this type of model gives the software much more accessibility to the state of hardware than is often the case when the software is run on the physical hardware.

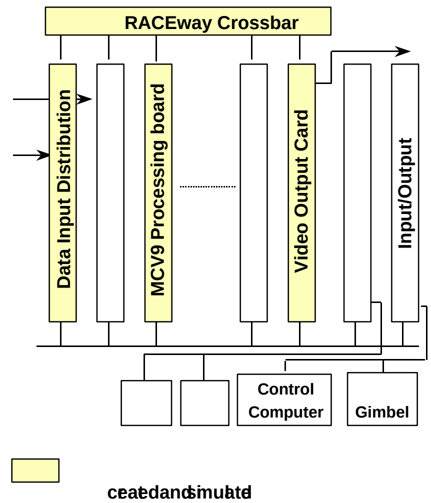
This slide shows that while increasing the modeling accuracy can improve on the capability for performing certain tasks, it will burden the simulation performance in terms of speed. Thus a right compromise is needed to tailor the requirements of the simulation and its level of fidelity and accuracy. A gate level model of an entire radar system would be unacceptable, however, early performance modeling of radar systems can be completed in minutes using performance models instead of gate level models.

Reinventing Electronic Design
Architecture Infrastructure

Case Study: Design IRST System Prototype

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Data Input Distribution card
- Sensor Interface card
- Video output card
- VME interface
- RACEway



, Fall 1995, pp. 9-21,

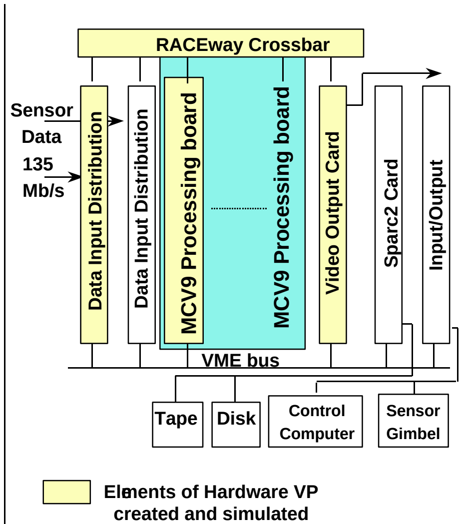
http://www.computer.org.

Case Study: MCV9 Processing Boards

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

RASSP E&F — SCRA • GT • UVA — Raytheon • UCinc • ADL

- **Hardware Elements**
  - **Data Input Distribution card**
  - **RS-170 Daughter card**
  - **Sensor Interface card**
  - **Video output card**
  - **MCV9 processing boards**
  - **VME interface**
  - **RACEway XBAR network**



 79

In the case study, the MCV9 board was virtually prototyped entirely in software, wherein actual software was executed on fully functional models for the hardware and the interconnect (RACEWAY and VME).

Methodology
RASSP
Reinventing
Architecture Infrastructure
DARPA ● Tri-Service

Case Study IRST:

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

- **Software Reset**
  - **Verify all boards could be reset via software**
  - **Specific registers should be configured with correct**
- **VME Register Test**
  - **SW written to read and write all registers on data**
  - **Known pattern placed in memory if tests are passed**

This and the following slides describe the types of test run at the system level to help verify that the system was implemented correctly.

Again, the reset test was the first to be done to verify that everything initializes correctly.

The VME register test was used to verify that registers in the data input and distribution card and video cards could be configured correctly. If the test was passed successfully, then a known pattern was written to memory.

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

Case Study IRST: System Tests (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

- **Floating Point RAM Test**
  - **Same as the previous RAM test, but different portion of RAM dedicated for FP**
- **Interrupt tests to check correct behavior of HW and SW responses**
  - **FIFO Overflow**
  - **Data Overflow**
  - **Beginning of Frame**
  - **End of Frame**

81

The floating point RAM test was similar to the previous RAM test, but a different memory was verified.

Interrupt tests were important to test both the HW and SW. The HW of the four designed boards could interrupt the processor based on whether their data and FIFO buffers were full. In this case the processor can take the appropriate measures to alleviate the problem. The HW also can send the processor information as to when the frame starts and stops. These tests usually took a long amount of time because some of these events happen much later in the timeline, as the next chart will show.

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

Case Study IRST: System Tests (Cont.)

- **RAM test**
  - **Read/Write portions of RAM on the video card via VME**
  - **128 locations written, then read back**
  - **Test disclosed major design error in VME logic of video and data distribution card**
  - **Misinterpretation of VME specification with respect to address lines A1 and A2; i.e., they were not used for decoding and therefore limited addressing to 32-bit locations**

The RAM test was used to read and write portions of memory on the video and the data and input distribution cards via the VME bus. A total of 128 locations were written and read back, and when this was tested an error was found on both cards. The designer misinterpreted the VME specification and did not use address lines A1 and A2 for decoding. This prohibited the use of addressing less than 32-bit locations, and the error was found during this test. This was a significant error that would have required a difficult fix later in the design cycle, but because no HW had been created at the time, the fix was done to the VHDL code. The new code was synthesized again with the fix.

Case Study MCV9: Test Process

- **Phase I: Initial Integration (Processing Element)**
  - **i860 <=> CE-ASIC <=> Memory**
- **Phase II:**
  - **PE <=> XBAR**
- **Phase III:**
  - **Multiple XBARs**
  - **Communications with boundaries**
    - **VME**
    - **RACEway**
  - **Added VME Driver and Interlink at this point**
  - **Multiple Processing Elements**

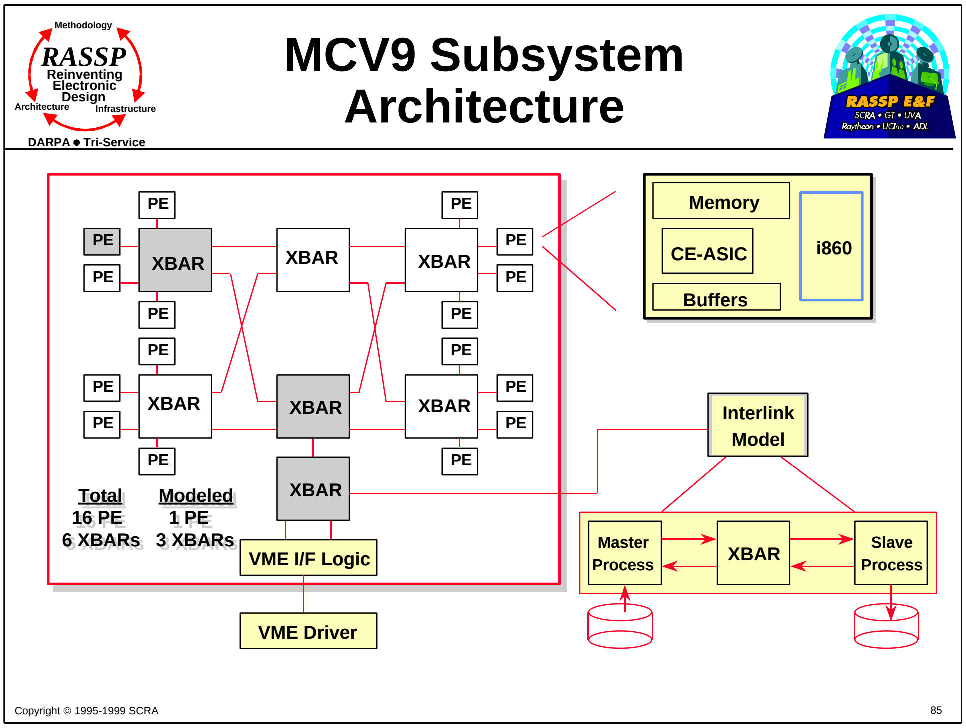
Various tests were used to verify the integration of the components. These were done in phases and were part of a test and integration plan. The first phase tested the processing element alone, which included the memory, the i860, and the CE-ASIC, along with some buffer registers. Phase II attached the processing element to a single XBAR and phase III connected multiple XBARs. Phase III also included tests to write to the interface of the MCV9 (VME and RACEway Interlink).

Case Study MCV9: Phase II/III Integration Tests

- **Tests at this stage**
  - **i860 <=> XBARs <=> RACEway <=> Interlink**
    - **Verify writing data to RACEway from the i860**
  - **i860 <=> XBARs <=> VME <=> VME Driver**
    - **Verify writing data to the VME interface**
  - **i860 <=> XBARs <=> i860**
    - **Verify multiprocessor communication**
- **VME used for passing control information**
- **RACEway used for passing video data**

The actual tests run at this phase included those listed above.

Control information is passed over the VME bus in the actual system architecture, and video data was passed over the RACEway interlink.



This figure shows more detail of what is contained on an MCV9 board. The shaded regions represent the items that were modeled in VHDL. The XBAR models were generated from lower level gate models of the components. The processing element was developed at the detailed behavioral level. In order to run code on the system, only one processing element was required. The control code resided on the processor and configured external hardware via the VME bus. Interrupt information from external hardware was also sent to the processor via the VME bus. When the input sensor buffers were full of data, they would notify the processor to configure a transfer to the internal memory the processing elements.

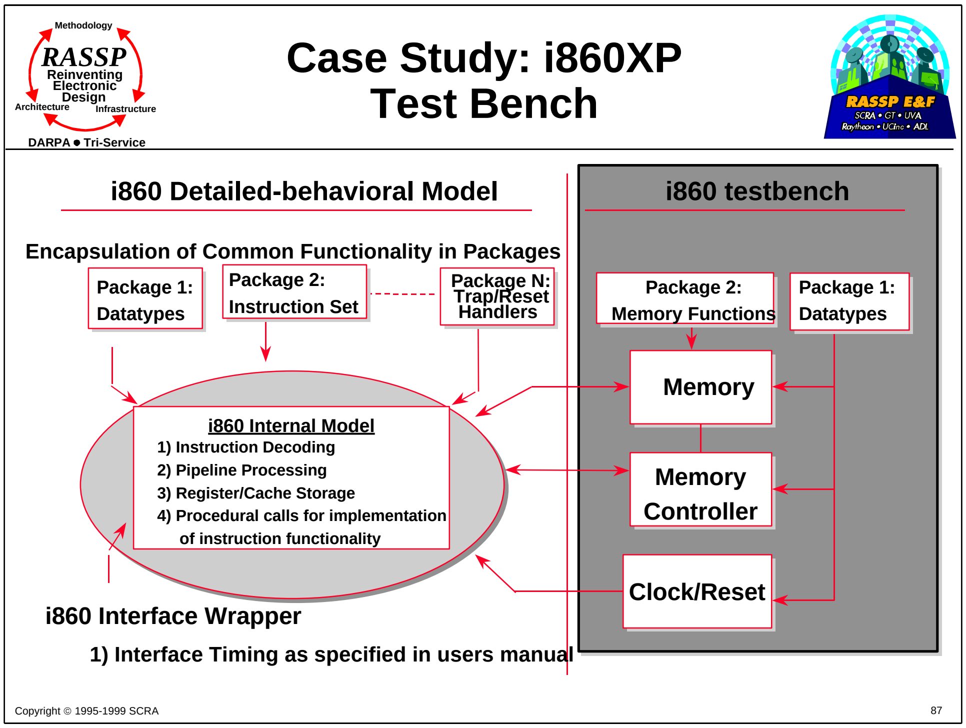
- i860 was developed from the data manual description
  - Clock Cycle accurate
  - Behavioral Description
- CE-ASIC and XBAR models developed by converting existing schematics
  - Translation tools from Mentor Graphics and Viewlogic
  - Not straight-forward and not what was expected in all cases
  - A configuration file was generated for entire subsystem

This slide describes how and at what level the VHDL models for the various component elements of the system were created.The i860XP model was created at the detailed behavioral level while the XBAR models were created from translation tools which used gate-level models. The entire was configured and tied together using a configuration file at the top-level.

MCV9 VHDL Creation

- **i860 was developed from the data manual description**
  - **Clock Cycle accurate**
  - **Behavioral Description**
- **CE-ASIC and XBAR models developed by converting existing schematics**
  - **Translation tools from Mentor Graphics and Viewlogic**
  - **Not straight-forward and not what was expected in all cases**
- **Configuration File for entire subsystem**

This slide describes the top level view of a board model.



This diagram outlines the i860XP component model. It is composed of the i860XP design unit and a testbench. The i860XP is represented as a fully functional model.

The internal model is at the behavioral level.

The interface model, which accurately models the timing information at the interface, has hooks to the internal model.

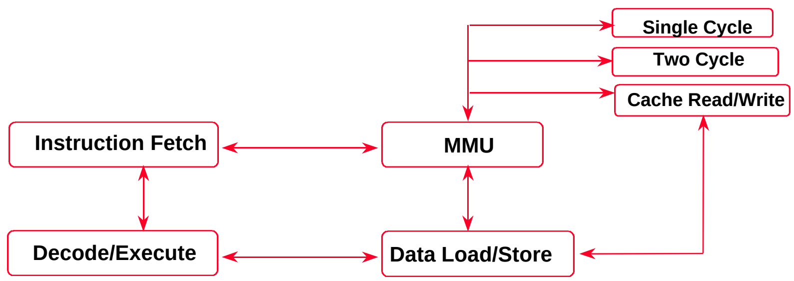
Packages used for functionality encapsulation:

- Datatype/Conversion
- Instruction Set Implementation
- Trap/Reset/Interrupt/Exception Handling
- IEEE Standard 754 Floating Point Math

The test bench consists of a memory, a memory controller, a clock and reset generator for synchronization, and assorted packages to encapsulate specific functionality related to each element. The test bench also contains the test program which is stored in files and read into memory as needed.

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

Case Study i860: Functionality Breakdown



- **Seven Processes**
  - **Needed to simulate concurrency of units**
  - **Approx. 200 instructions/sec**
- **Initially single process (ISS)**
  - **Interface behavior not needed**
  - **Approx. 2000 instructions/sec**

Given the various processor elements, the first step was to break down its functionality into specific processes. The initial attempt was to place all the functionality into a single process similar to instruction set simulator type models. This permitted faster running models, but could not capture all the concurrency issues of the processor. For example, if a cache miss occurred, the processor would need to go to external memory for data. In this case we do not want to stop the main process from executing what was already in, for example, one of the floating point pipelines, to wait for the data to arrive. Because multiple processes were required, the above 7 were chosen to represent the behavior of the device. The next slide lists the type of functionality in each process.

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

Case Study MCV9: Subsystem-level Test

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Objectives of Subsystem Testing**
  - **Does each component model behave correctly stand-alone?**
  - **Is the handshaking protocol between components or buses functioning?**
  - **Are the data rates between components within the specified limits?**
  - **Does the prototype provide a high level of confidence for HW prototyping?**

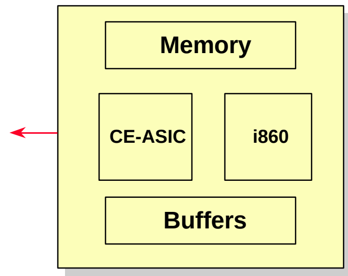
89

When doing subsystem integration, a plan was required to determine the objectives expected from doing this simulation and when to end. The objectives are listed above. At this level of abstraction the main information to be gathered included whether the components interface correctly and if the data rates between components are at the rate required to meet performance objectives. Lastly, to end simulation runs, one needs to determine how much simulation is enough, and this is based on the degree of confidence the designers have as to whether the actual HW will work based on the simulations. Because the MCV9 is a COTS part, there was a high degree of confidence from the beginning, and detailed simulations were not required. It was sufficient to have the units talk together correctly, and the data was sent across the crossbar network in the correct time.

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

Case Study MCV9: Phase I Integration Objectives

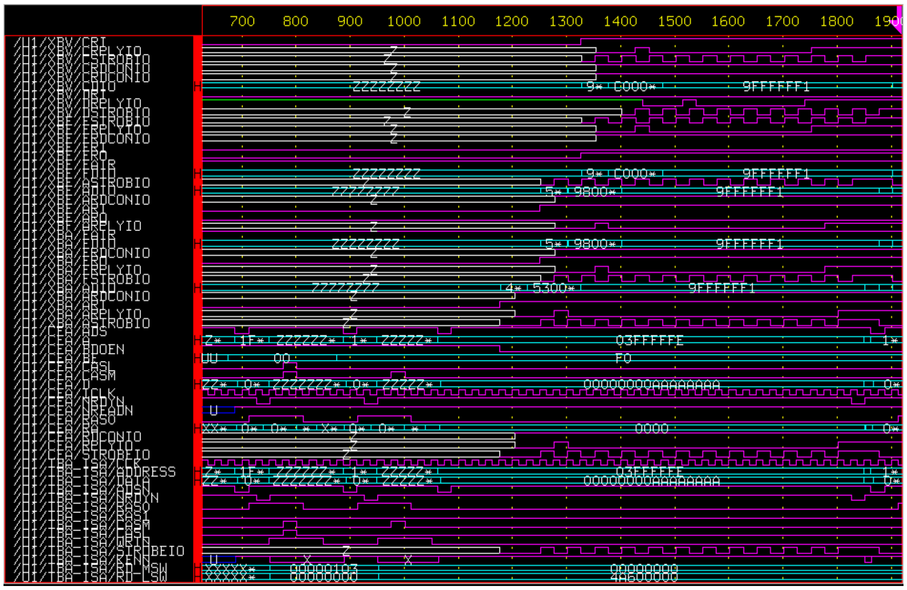
- **Tests at this stage**
  - **Reset**
    - **Did anything reset to an unexpected state?**
  - **CE-ASIC registers**
    - **Can the i860 set key registers in the CE-ASIC?**
  - **Reading/Writing to CE-ASIC**
    - **Is the handshaking logic working between the i860 and the CE-ASIC?**
  - **Memory**
    - **Can we read and write data to the memory?**



 90

The phase I tests run are listed above. Their intent was to test the integration in the processing element of the MCV9. The first test performed was a reset test to verify that all the elements came up in expected states after reset. Once this was verified, then various register tests were implemented to test the ability of the processor to set registers in the CE-ASIC. At the same time, the handshaking protocol was verified between the i860 and the CE-ASIC. Finally, reads and writes to memory were tested to verify the interface connection was correct.

Case Study MCV9: Timing Diagram of RACEway Write



The diagram shows the communication from a processing element across the RACEWAY communications channel to another processing element on another board. This level of detail is visible without any hardware emulation and only using VHDL and executable software running on the processor models.

For futher details see Madisetti & Egolf (IEEE Micro, Fall 1995, pp. 9-21)

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

Case Study MCV9: Phase II/III Integration Objectives

RASSP E&F — SCRA • GT • UVA — Raytheon • UCinc • ADL

- Interface a single XBAR to the PE, then multiple XBARs
- Check Communications at boundaries
- Two major interfaces
  - VME
    - VME bus model added to test bench
  - RACEway
    - Interlink model added to test bench
- Full MCV9 16 processor instantiation
  - Data written between processors

 92

Once it was guaranteed that the processing element was functioning properly, a XBAR, and then multiple XBARs, were added to the model. The handshaking protocol was again verified, and communications were checked to the subsystem boundaries. These included the two major interfaces: the VME and the RACEway interlink. When it was verified that this communication was working properly, a full instantiation of 16 processors was tested on the MCV9. In this case data was written between processors to make sure the routing was working correctly.

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

Case Study MCV9: Phase II/III Integration Tests

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

- **Tests at this stage**
  - **i860 <=> XBARs <=> RACEway <=> Interlink**
    - **Verify writing data to RACEway from the i860**
  - **i860 <=> XBARs <=> VME <=> VME Driver**
    - **Verify writing data to the VME interface**
  - **i860 <=> XBARs <=> i860**
    - **Verify multiprocessor communication**
- **VME used for passing control information**
- **RACEway used for passing video data**

 93

The actual tests run at this phase included those listed above.

Control information is passed over the VME bus in the actual system architecture, and video data was passed over the RACEway interlink.

Case Study MCV9: Simulation Results

Test Case	Internal Simulation Time	Instr/sec.
MCV9 to Interlink	1200 ns	7.5
MCV9 to VME Driver	3000 ns	5.5

- **Application code requires significant simulation time**
- **Diagnostic and test code may be less compute intensive**
- **Test and Diagnostic can represent the majority of code**

Simulation results were collected for two of the tests. The two tests included the i860-to-interlink data writes and the i860-to-VME writes.

The number of instructions/sec executed with all the additional models added to the subsystem prototype has decreased significantly. This prohibited the running of application code on the virtual prototype.

Application code was not run on the prototype because it required too many computing resources, but test and diagnostic code is a viable candidate for code running on a virtual prototype. In some systems, the test and diagnostic code can represent the majority of the code.

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

Case Study IRST: Simulation Results

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Events happening far into simulation timeline not able to be seen
- Large amount of disk space required for results database storage
- Test plan must be established early in the prototyping effort to account for long simulations
- Methods must be devised to stop long simulations if bad results occur early in the simulation

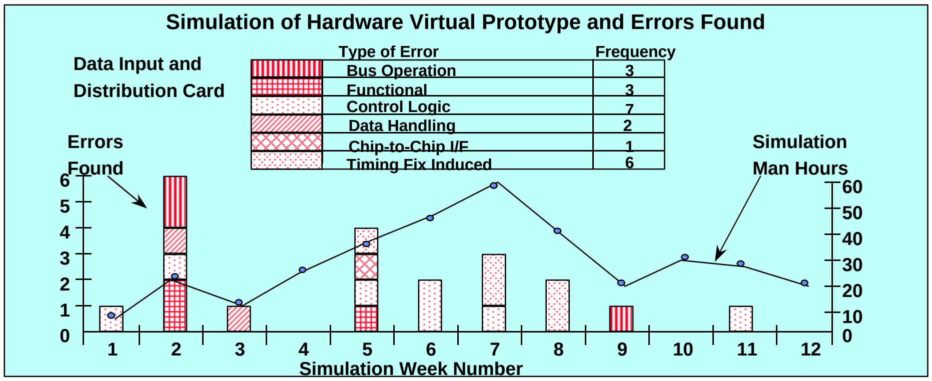
Class of Machine: Sparc10 / 128 Meg. RAM / 1.2G Disk
Signals archived: 1,048
Components: 13,257

Simulation Time (msec)	CPU Time (hours)	Disk Space (MB)
1.0	3 - 4	10 - 20
2.0	5 - 6	60 - 80
3.0	7 - 9	90 - 110
4.0	10 - 11	130 150
5.0	11 - 13	170 - 190

As can be seen from the chart on the right hand side, there is a near-linear relationship between simulation time and amount of CPU time to run the test code. The same applies to the amount of memory required to save the results database.

From this information, it is obviously critical that a test plan be devised at the beginning of the modeling effort to account for these long simulations and to institute methods to stop long runs when an error occurs early in the simulation. The test plan must also address the issue of how much simulation is satisfactory before acceptance of the HW design and HW prototyping can begin.

Case Study IRST: HW/SW Integration



- Board Fabrication — Avg. = 5 days
- Board Assembly — Avg. = 3 days
- HW Checkout — Avg. = 3 days
- HW/SW Driver Integ. — Avg. = 12 days

Approx. 23 days for Initial integration

 96

This slide illustrates the types of errors found during early integration and testing using the detailed behavioral virtual prototype in the design process. The errors were found on the data input and distribution card when software was being executed on the processing elements of the MCV9 board. From the figure, it is seen that the errors were tracked over the 12 week period of testing and as the errors start to decrease, it was determined that the board could go to fabrication with a high degree of confidence in correctness. After fabrication, the final hardware was integrated in 23 days and there were no bugs in the digital hardware that was tested using the virtual prototyping approach.

Detailed Section Outline

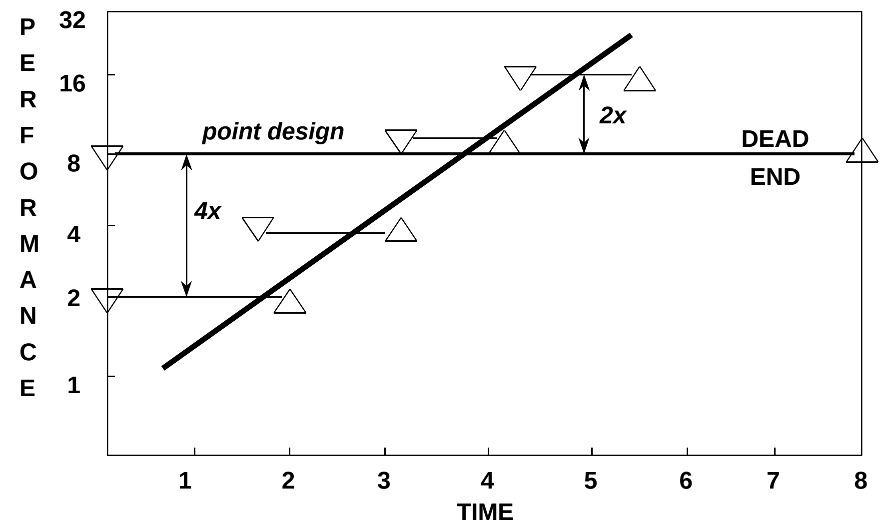
- Fully Functional Modeling
  - Description
  - Case Study
- **Model Year Architecture**
- Reuse

Section Outline

- **The Model Year Architecture (MYA) framework**
  - **MYA within the design process**
  - **Components of the MYA**
  - **The Standard Virtual Interface (SVI) and reuse**
  - **Design for test architectures**
  - **Software architectures**

Model Year Architecture (MYA) is a framework for reuse that provides a structured approach to ensure that designs incorporate the features required to promote upgradability. The basic elements that comprise the MYA are the Functional Architecture, Encapsulated Library Components and Design Guidelines and Constraints. Synergism between the Model Year Architecture Framework and the RASSP Methodology is required, as all areas of the methodology, including architecture development, hardware/software codesign, reuse library management, hardware synthesis, target software generation, and design for test are impacted by the MYA Framework. The Functional Architecture defines the necessary components, and their interfaces, to ensure that the design is upgradable and facilitates technology insertion. It is a starting point for developing solutions for an application-specific set of problems, not a detailed instantiation of an architecture.

Model Year Chart



Three keys aspects of the RASSP design methodology that support the "Model-Year Approach":

Re-Use Library

Architectural-level reuse library element class definition.

Library management reuse library element generation/validation.

Virtual prototyping

Virtual testbench analysis of each model year system.

Use of Standards

Common bus and system architectures, code and data structures, and design protocols.

Model Year Architecture

Model year architecture goals:

- **Design current generation with a view to change and upgrade on a regular basis**
- **Develop the design as a series of incremental (model year) stages (as opposed to a single design effort spanning a multi-year design schedule) that can incorporate emerging technologies as they become available during the successive model years**

The key is to make provisions in the early model years to make including new technologies easy in the later years.

The model year architectural approach should adhere to the following principles:

• Be open, promoting hardware/software upgradability and reusability in other applications.

• Utilize emerging state-of-the-art commercial technology whenever possible.

• Support a range of applications to maintain low, non-recurring engineering costs.

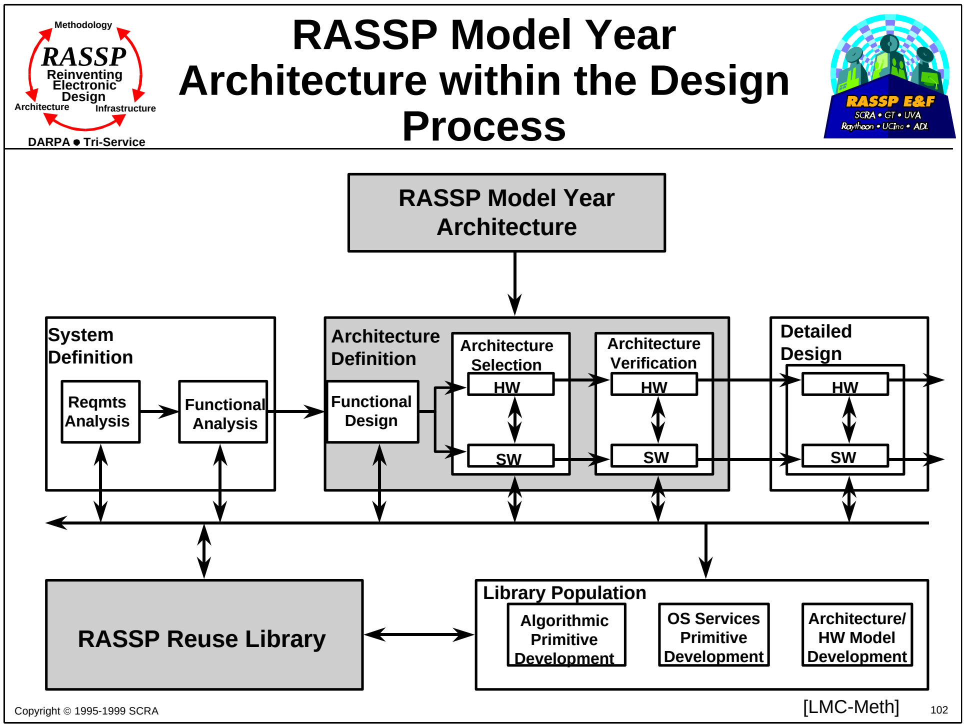
Model Year Architecture

- **Objectives**
  - **Develop framework for signal processor architectures**
  - **Support sufficient model-year upgrades by minimizing hardware/software breakage**
  - **Develop model-year instantiations to support benchmarks and demonstrations**
  - **Promote design upgrades and reuse via standardized, open interfaces while leveraging commercial technology**
  - **Support scalability, heterogeneity, modular software, life cycle support, testability, and system retrofit**

As technology is evolving faster than systems can be developed, the concept of Model Year Architecture allows the incorporation of new technologies to be inserted into the design as they appear.

The objective of the Model Year Architecture (MYA) is to develop a framework for signal processor architectures. The MYA should address the following issues:

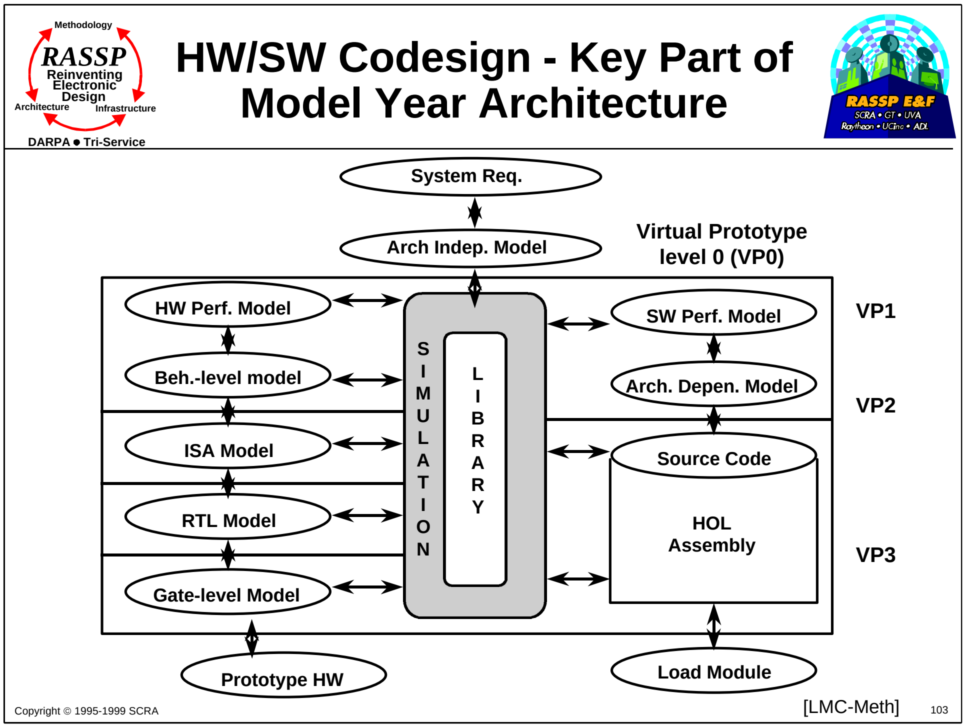
- Contribute to the 4X reduction in design cycle time required by RASSP
- Provide life cycle support
- Provide scalability to support changing mission scenarios and different deployment environments
- Support heterogeneity in the design process by providing cost effective implementations of functions with a wide range of performance requirements
- Provide flexible interfaces to a wide range of subsystems
- Utilize modular software in the form of reusable components and support upgrades to operating systems, services, and libraries
- Support hardware upgrades
- Provide for testability in the design process and detect and isolate faults with high probability
- Support for RASSP signal processor retrofit into non-RASSP (legacy) systems
- HW and SW elements within the library of components are encapsulated by functional wrappers, which add a level of abstraction to hide implementation details and facilitate efficient technology insertion



To dramatically improve the process by which complex digital systems are specified, designed, documented, manufactured and supported requires a signal processing design methodology that recognizes a number of application domains. A key element to implement this methodology is a Model Year Architecture approach that adheres to a specific set of principles which include:

- The architectures must be open to promote HW/SW upgradability and reusability in other applications.
- The architectures must use emerging, state-of-the-art commercial technology whenever possible.
- The architectures must support a wide range of applications to maintain low non-recurring engineering (NRE) costs.
- The architectures must facilitate continuous product improvement and substantial life-cycle-cost (LCC) savings in fielded system upgrades.

The RASSP Model Year Architecture(s) (MYA) must be supported by the necessary library models to facilitate trade-offs and optimizations for specific applications. Reusable HW and SW libraries facilitate growth and enhancement in direct support of the RASSP model year concept. The notion of model year upgrades is embodied in the reuse libraries and the methodology for their use. As technology advances, new architectural elements may be included in the library. Rapid insertion of a new element into an existing, RASSP-generated design is the goal of the Model Year concept.



The RASSP design process is based on true HW/SW codesign and is no longer partitioned by discipline (e.g. HW and SW), but rather by levels of abstraction represented in the system, architecture, and detailed design processes. The above figure shows the RASSP methodology as a library-based process that transitions from architecture independence in the systems design process to architecture dependence in the architecture process.

Various levels of virtual prototypes are generated throughout the design process. The first is output from the systems process, where an executable specification is generated, the architecture process generates two more with increasing detail and verification. The final prototype is created before HW/SW sign-off and full system verification is done at the RTL and gate levels with application and test SW running on the prototype.

Components of Model Year Architecture

- **Functional architecture**
  - **Provides a starting point for developing application-specific architectures**
- **Encapsulated library components**
  - **Provides component interoperability and upgradability at the architectural level**
- **Modular software architecture**
  - **Supports upgrades for operating system kernels, external services, and application libraries**
- **Open interface standards**
  - **Ensure wide availability of commercial components and support**
- **Design guidelines and constraints**

 104

The basic elements of a MYA are listed above.

The functional architecture defines the necessary components and the manner in which their interfaces must be defined to ensure that the design is upgradable and facilitates technology insertion. As such, the functional architecture is a starting point for developing solutions for an application-specific set of problems, not a detailed instantiation of an architecture.

An important aspect of the functional architecture is that application-specific realizations of a signal processor are embodied in the proper definition and use of encapsulated library elements. Encapsulation refers to additional structure added to otherwise raw library elements to support the functional architecture and ensure library element interoperability and technology independence.

A modular software architecture simplifies the development of high-performance, real-time DSP applications allowing the developers to easily describe, implement, and control signal processing applications for multiprocessor implementations. It supports upgrades for operating system kernels, external services, and application libraries.

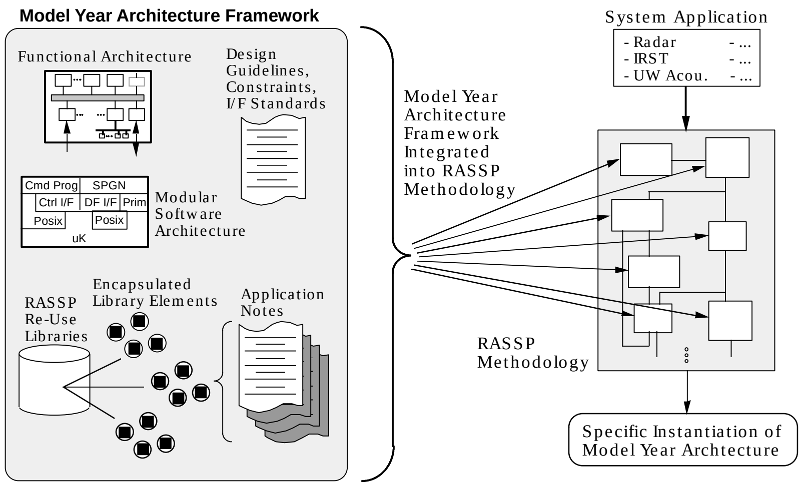
Open interface standards are used to help ensure interoperability between components and ensure a wide availability of commercial components and support.

Design guidelines and constraints are provided for general architectural development, such as how to use the functional architecture framework, use of encapsulated libraries and procedures and templates to encapsulate new library components.

[LMC-ATL][LMC-MYA]

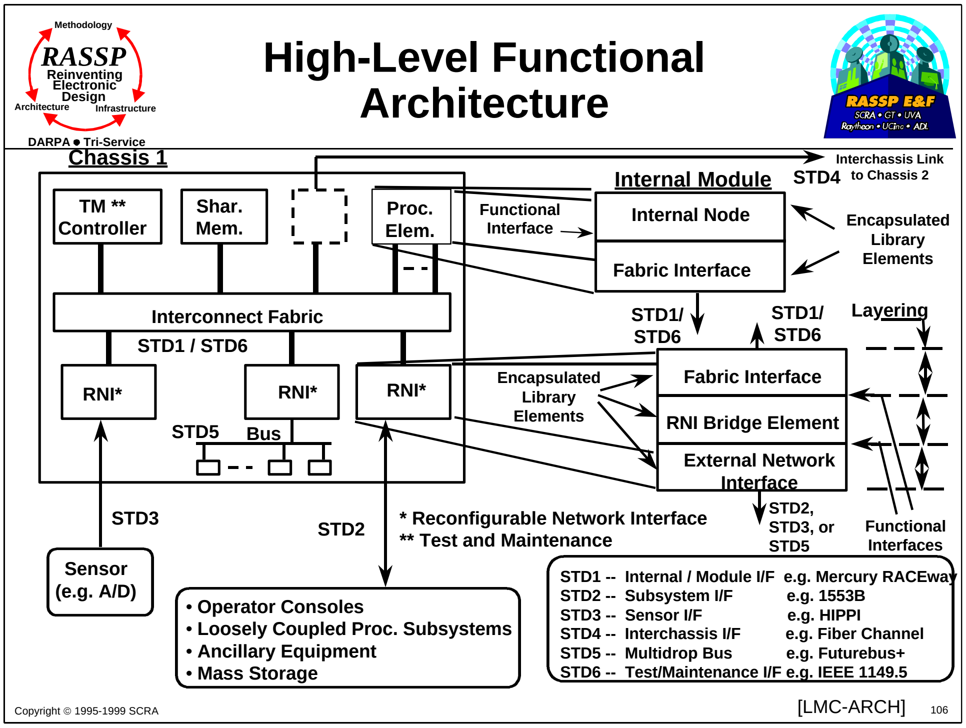
RASSP Reinventing Electronic Design
DARPA ● Tri-Service

Model Year Architecture Approach



[LMC-ATL] 105

Model-Year architecture provides a framework for reuse and reuse-affordance technology upgrades.



The functional architecture defines the necessary components and the manner in which their interfaces must be defined to ensure that the design is upgradable and facilitates technology insertion. The functional architecture is the starting point for developing solutions for an application-specific set of problems, not a detailed instantiation of an architecture. The functional architecture DOES NOT specify the topology or configuration of the signal processing architecture.

The functional architecture specifies a high-level framework for launching application-specific architecture development. Architecture-level reuse element classes are provided. Open interface candidates for the interconnect fabric, sensor, and interchassis interfaces are provided for selection. The functional architecture also specifies the test methodology to be used for design.

The STDx demarcations illustrate the types of interfaces found in various portions of the functional architecture.

The Reconfigurable Network Interface (RNI) is divided into three logical elements: 1) Fabric interface, 2) External network interface, and 3) Bridge element. The fabric and external interfaces implement the specific protocols to the elements being interconnected, for example a High-speed Parallel Port Interface (HIPPI) could be used for the external interface and a VME interface can be used for the fabric interface. The bridge element, which typically consists of a buffer memory and a controller implemented via custom logic (e.g. FPGA, ASIC) or a programmable processor, performs the actual bridging function. The buffer memory facilitates asynchronous coupling and flow control between the two networks, while the controller coordinates data transfers. The three logical elements of the RNI are implemented as encapsulated library elements that serve to isolate changes resulting from upgrades. For example, the VME interface can be replaced with an encapsulated SCI interface.

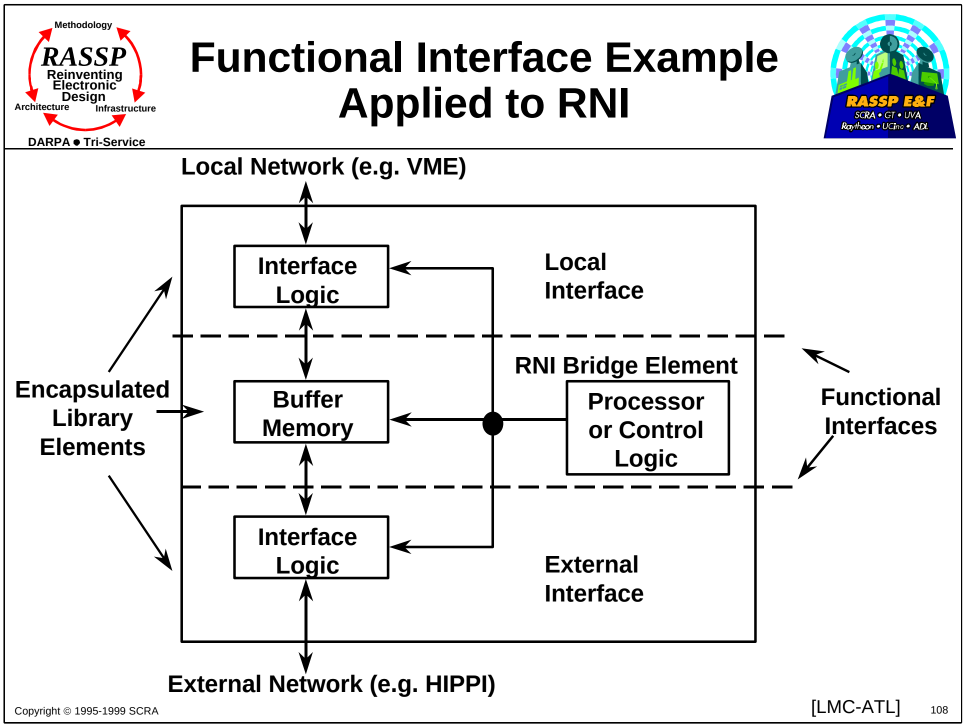
The processing element is also encapsulated so links to the internal interconnect fabric is made easier, reusable and provides a better route to upgradability.

Interface Approach

- Use layered approach
  - Decompose architecture into smaller, manageable, and reusable parts
- Define standard functional interfaces, not physical interfaces
  - Technology independence to support model year upgrades
- Provide guidelines for a Standard Virtual Interface (SVI)
  - General interface to support different communication paradigms
  - Adds additional layer to hardware interfacing
- Use standard Application Programming Interface (API)
  - Data flow graph approach similar to PGM
  - Isolate application SW from underlying operating system implementation

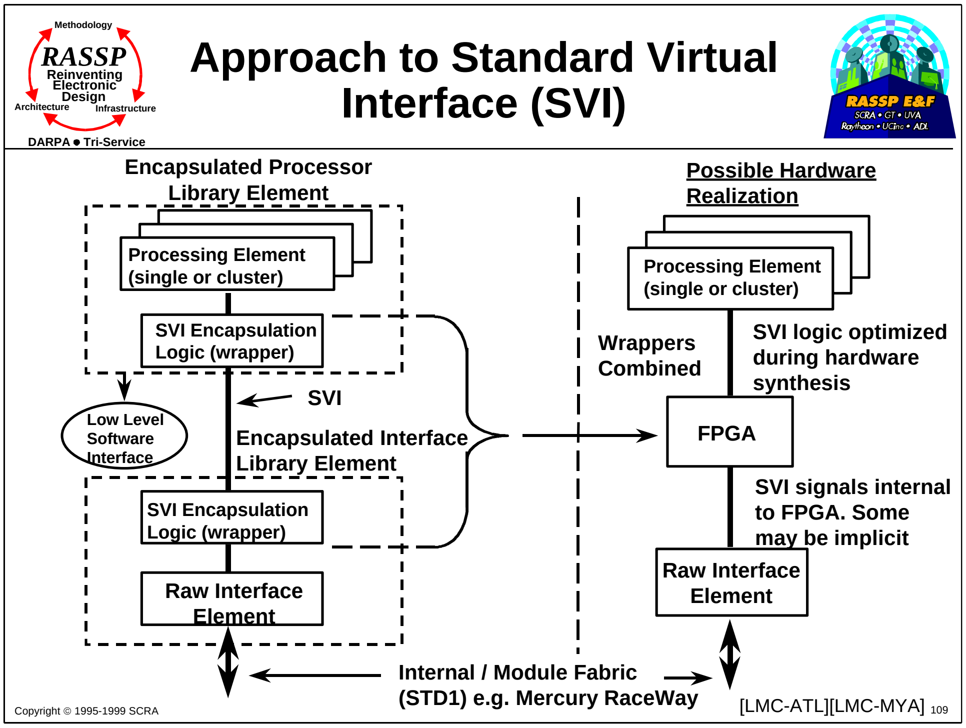
 107

A layered approach can be used for handling the interfacing between components. This decomposes the architecture into smaller, manageable, and reusable parts. A standard functional interface was defined supporting technology independence and model year upgrades. The interface is implemented using a Standard Virtual Interface (SVI) which is general enough to support different communication paradigms and adds an additional layer to the hardware interfacing. SVI will be discussed in more detail in the following slides. An Application Programming Interface (API) is used to isolate the SW from the underlying operating system implementation.



The above diagram illustrates an application of a functional interface at the hardware level for a construct called an Reconfigurable Network Interface (RNI). The RNI is divided into three logical elements: 1) local interface, 2) external interface, and 3) bridge element. The local and external interfaces implement the specific protocols to the elements being interconnected, in this example a HIgh speed Parallel Port Interface (HIPPI) and VME interface. The bridge element, which typically consists of a buffer memory and a controller implemented via custom logic (e.g. FPGA, ASIC) or a programmable processor, performs the actual bridging function. The buffer memory facilitates asynchronous coupling and flow control between the two networks, while the controller coordinates data transfers.

The three logical elements of the RNI are implemented as encapsulated library elements that serve to isolate changes resulting from upgrades. For example, the VME interface could be replaced by another encapsulated interface, such as SCI, with little or no impact on the HIPPI HW and SW.



SVI encapsulates library elements to support reusability and rapid upgradability. The interconnection of library elements is done by connecting their SVIs. A protocol is defined for the SVI to SVI interface. Each library element needs the SVI to operate in this environment. A possible hardware realization is shown above. The SVI interface is implemented on an FPGA, or an equivalent technology, using optimized hardware synthesis tools.

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

Guidelines for Using SVI

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Use SVI only down to the smallest desirable upgradable LRM in the system
- If the line replaceable module (LRM) is a board: SVI should be associated with the board-level interface only and not for any intra-board interfaces
- If the LRM is an MCM or a daughter card: SVI should be associated with the MCM-level interface only and not for any intra-MCM interfaces
- The final choice for using SVI at an architecturally finer-grained level depends on the relative HW overhead incurred by SVI for a given scenario

SVI can be used at any encapsulation level (LRM, MCM, component), but should be used where it makes the most sense. Considerations of HW overhead and reusable design elements should be taken into account.

Applications of SVI

- **Internal modules**
  - **Use for processing nodes and shared memory nodes**
  - **Use between internal node and node-to-interconnect interface**
  - **Allows "plug and play" interoperability between internal nodes and node-to-fabric interfaces**
- **Reconfigurable Network Interface (RNI)**
  - **Implements system-level interfaces: sensors, loosely coupled processor subsystems, operator consoles, etc.**
  - **Implements bridge between internal interconnect fabric and particular system-level interfaces**
  - **Contains three logical components: fabric interface, external network interface, and RNI bridge element**

SVI can be applied at the module level to encapsulate processing and shared memory nodes, at interconnect boundaries to allow for plug and play interoperability between internal and interface elements, etc.

The choice of encapsulation depends on issues of supportability, design overhead, etc.

Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

Implications of Layering

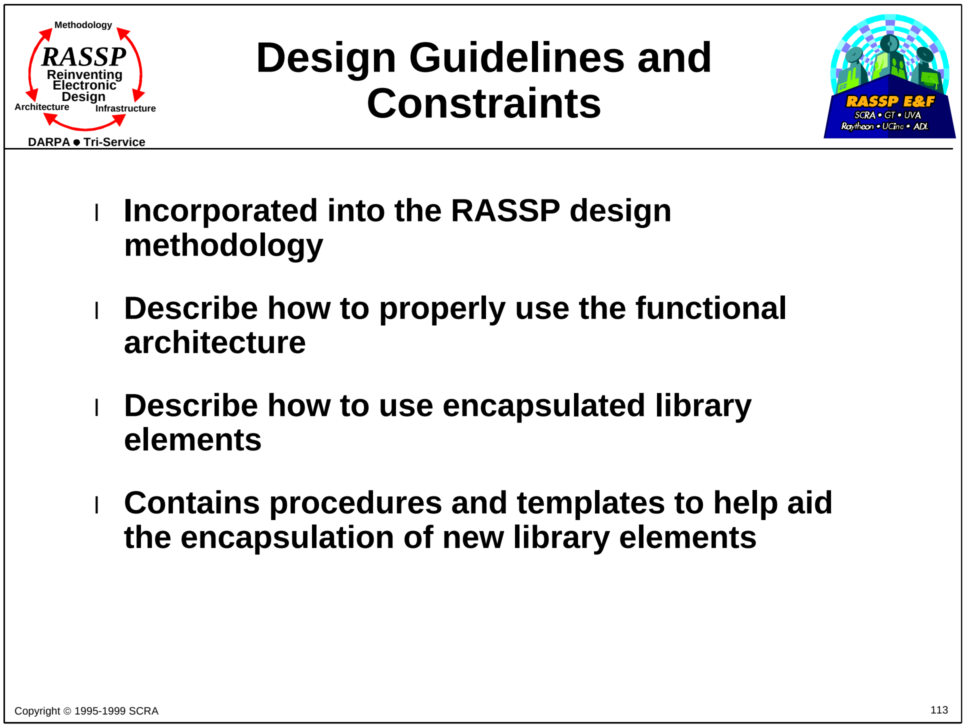
RASSP E&F

- **Can cause performance penalties and unacceptable hardware or software growth**
- **Resolution**
  - **Trade off performance vs. functionality**
  - **Overhead reduction techniques -> SVI**
- **Use layering judiciously; only isolate important architectural elements subject to upgrades/ technology insertion**
- **Some layering overhead must be accepted**
  - **Tradeoff to realize greater benefits of design/life cycle and cost reduction**

 112

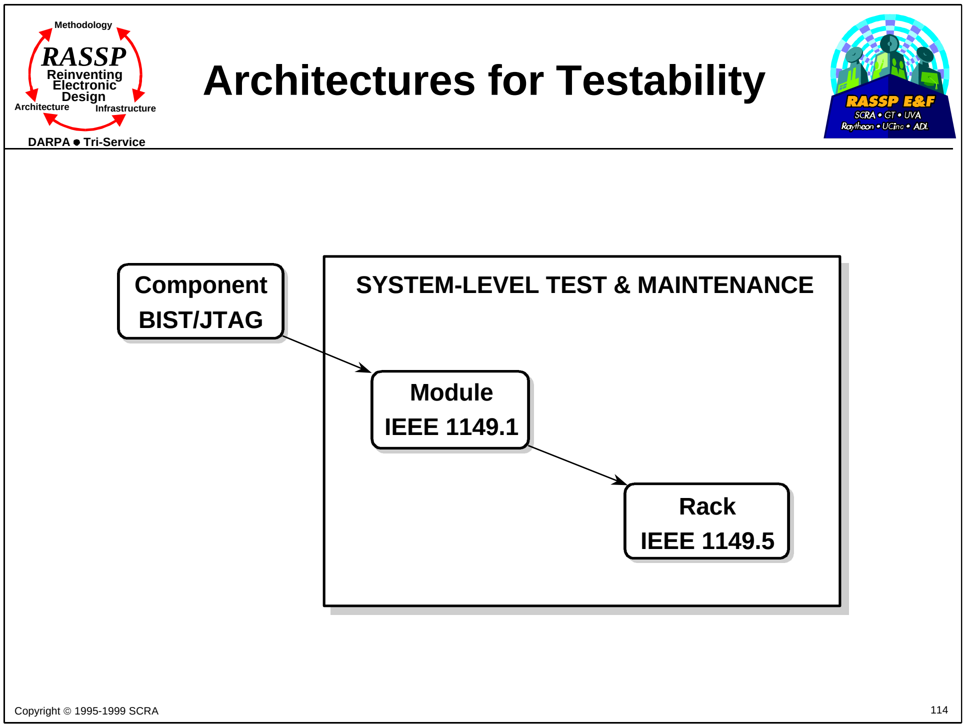
Layering can cause performance penalties due to the additional HW overhead. This can be acceptable if the layering is chosen judiciously and only important architectural elements are isolated where possible technology insertion can occur.

[LMC-ATL][LMC-MYA]



As part of the MYA framework, an important feature is the capture of guidelines of various workflows in the design process and incorporate them into the RASSP methodology. Guidelines are also described for encapsulating new elements to be placed in the design library.

[LMC-ATL]



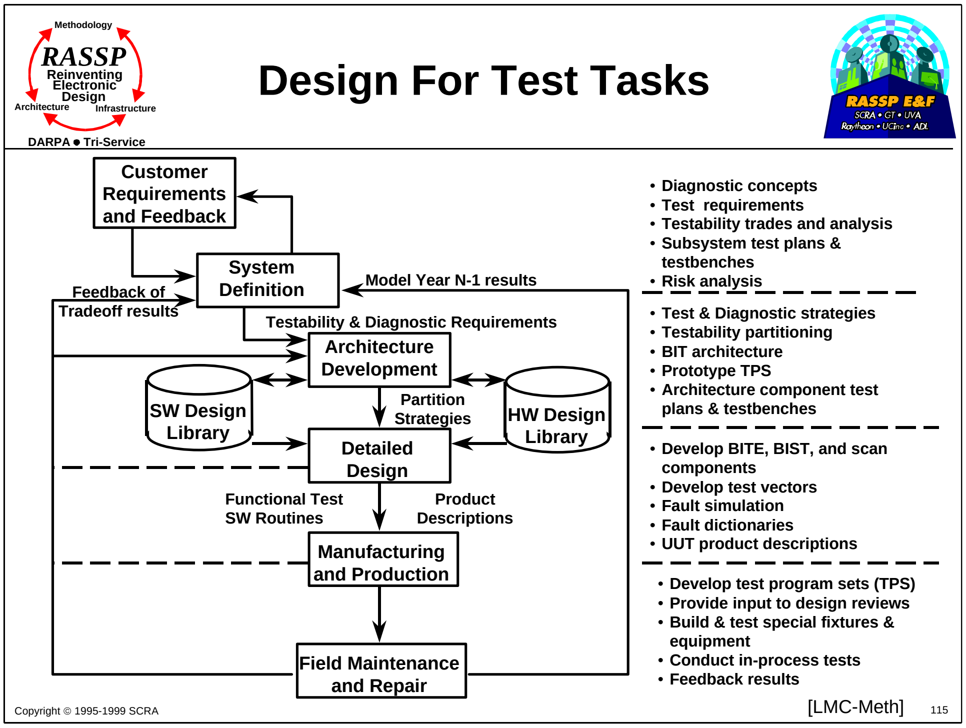
To provide an integrated diagnostic capability, a structured test approach is required for the various levels of system integration: component, module and box (rack).

Component: High degree of fault coverage (>95%) should be provided. BIST should conform to the IEEE 1149.1 standard (JTAG). Many IC vendors now provide for it.

Module: IEEE 1149.1 boundary scan architecture is used to detect interconnect faults between components. Modules are designed with built-in-test (BIT) to detect, diagnose and isolate module faults. This is usually controlled by a BIT controller.

Rack: A test and maintenance (TM) controller manages system-level testing, including the initiation of BIT for each of the modules. IEEE 1149.5 proposes a TM bus standard.

System Test requirements may vary significantly based on the application.



Designers of complex systems cannot afford to postpone test considerations until the final stages of design and still deliver a quality product. Testable systems are not a natural product of a design team unless BIT and scan features are included up front and knitted together seamlessly throughout the system hierarchy.

To ensure consistency between levels of the design hierarchy, a system-level test architecture and strategy must be developed and passed down to each level. The DFT methodology uses the hierarchical partitioning to manage test development complexity and to provide solutions to the incorporation of COTS components.

The RASSP design process is shown above with specific information flow and activities relative for design for test. A prime goal of the RASSP methodology is to eliminate design modification efforts late in the design cycle, including those to correct testability problems. VHDL and WAVES are used, as appropriate, throughout the methodology to capture and refine test and DFT-related information.

Test Architecture

- Required as an integral part of MYA: Formal structure required to ensure testable design
- Use of standard test interfaces as required augmentations to signal/control interface to chips, module, and systems
- Test architecture parallels system architecture hierarchy
  - Processor system
  - Chassis
  - Sub-chassis/functional group
  - Printed circuit board
  - Line replaceable module (LRM)
  - Multichip assembly
  - Chip
  - Functional or logic block

 116

The test architecture is an important part of the MYA. Standard test interfaces should be augmented to the signal and control interfaces to chips, modules and subsystems.

The test architecture hierarchy should parallel the system architecture hierarchy incorporating elements at the system level, chassis level, all the way down to the functional or logic block.

[LMC-ATL][LMC-MYA]

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

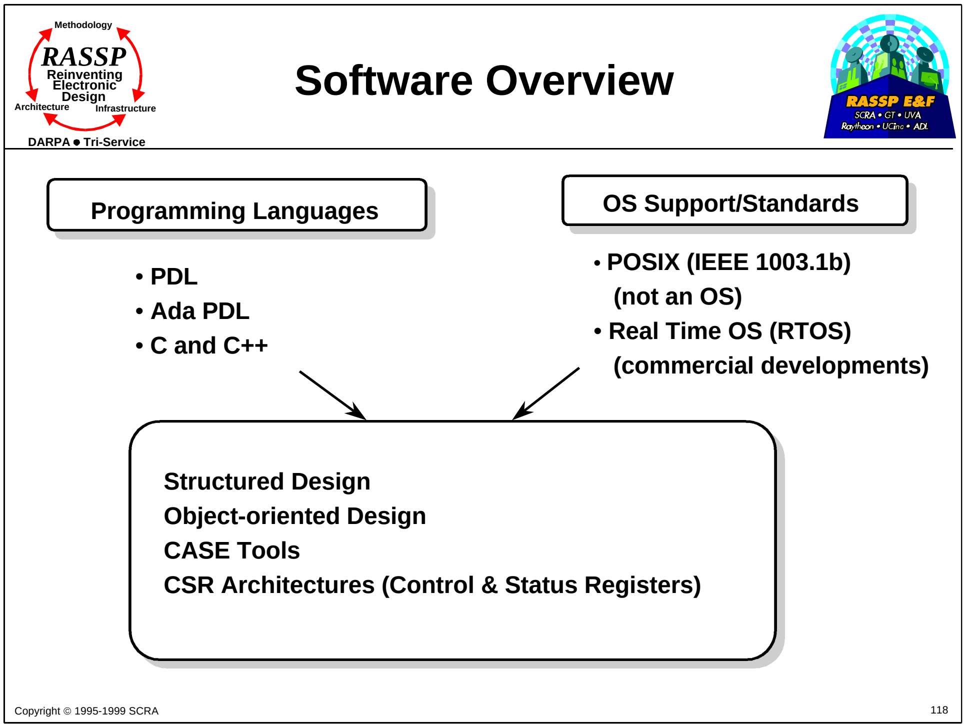
Test Architecture (Cont.)

- **Test architecture implemented by hierarchy of test and maintenance controllers (TMCs)**
  - **Participate in all TM activities**
  - **Communicate via standard test interfaces buses**
- **TMC responsibilities**
  - **Interface with master TMC (at highest level of hierarchy)**
    - **Receive test instruction**
    - **Receive test data and control**
    - **Send test and status results**
  - **Internal responsibilities**
    - **Execute/supervise testing of components within its scope**
    - **Collect results and compile status reports**
    - **Send test instructions and data to subordinate TMCs (if any)**

 117

Test and maintenance controllers (TMCs) should be used to implement the hierarchy and should communicate via standard test interface buses. The test and maintenance controllers have the responsibility to interface with the master TMC, collect results and compile status reports.

[LMC-ATL][LMC-MYA]



Attributes of software are considered **architectural** when they express relationships between HW and SW that contribute to long term capacity for change. They are considered **design** when they are implementation specific.

The SW architecture must make provisions for several levels of control and task management. Open systems protocols should be considered. The architecture also must provide for an orderly flow of data throughout the system.

Operating System: An open systems approach should be selected for greater resistance to system obsolescence. POSIX provides for standard interfaces. They are called the Operating System Interface (OSI) and the Application Program Interface (API). Use of the POSIX standards should allow SW to be portable across similar platforms.

Programming Language: PDL (Program Design Language) is a mixture of language statement and control structures. It has the following characteristics:

- States design in a easily read fashion.
- It allows concentration on the design logic rather than implementation details.
- Documentation can be done concurrently.
- It is convertible to a high order language (HOL).

Ada is the official language of choice for large complex SW projects of the U.S. Govt. Ada 95 provides for object-oriented features. C and C++ code can be used when COTS technology is specified for use.

Structured Design: A SW development methodology that follows a hierarchical structure of SW module development and test.

Object-oriented Design: Results in a more modular design. There are three phases to this approach. One, Object Oriented Analysis (OOA), two, Object Oriented Design (OOD), and last, Object Oriented Programming (OOP). OOA and OOD are embedded in the CASE tools such as Cadre's *Teamwork,* and IDE's *Software Through Pictures.*

CSR (Control and Status Registers) architectures can be used to identify the module, select a working subset of its performance capabilities, enable BIST, and record the health status history. IEEE 1212-1991 specifies a standard CSR architecture.

Software Architecture Process Goals

- **Formalize reuse**
- **Emphasize DFG-driven autocode generation**
- **Focus on three major areas of SW functionality**
  - **Algorithm, as specified by a flow graph**
  - **Scheduling, communication, execution, as specified by mapping a graph to a specific architecture**
  - **General command/control software**
- **RASSP is attempting to automate the first two**
- **Use CASE-based code development, documentation, and verification for the general command/control software**

 119

This slide presents a list of the SW architecture process goals desired by the RASSP process. These include a formalized approach to reuse, DFG-driven autocode generation for application code, CASE-based code development for general command and control software when autocode generation is not available.

[LMC-Meth]

Software Architecture Requirements

- Support a methodology that simplifies high-performance, real-time DSP application software development
- Provides easy description, implementation, and control execution of signal processing algorithms
- Supports application development in a platform independent fashion
- Provides predictable deterministic response to all provided services
- Supports upgrades of operating system kernels, external services, and application libraries
- Supports hardware upgrades via hardware-specific software modules

 120

The requirements of the SW architecture include those listed above.

Support should be included that simplifies high-performance real-time DSP application SW development. The SW architecture should provide predictable responses to provided services and easy description, implementation, and control execution of signal processing algorithms. The architecture should support HW upgrades, OS kernel upgrades, and application development in a platform independent fashion.

[LMC-ATL][LMC-MYA]

Software Architecture

- Layered architecture to support model year concept
- Uses commercial microkernel technology to provide underlying services to support high-level application programming interface (API)
- RASSP Run-Time System (RRTS) builds on the microkernel services to provide higher level services to control and execute applications on multiple processors
- RRTS support to implement required services is external to microkernel
- Scheduling and execution paradigms being re-defined for RASSP: more distributed

The approach used on RASSP to implement the SW architecture is listed above.

A layered approach is used to support the MYA concept where the replacement of a specific processor and its microkernel would maintain the same API so applications developed for one processor need not be changed when porting it to a new system.

The RASSP run-time system (RRTS) is built on the microkernel to provide higher-level services to control and execute applications on multi-processor systems.

[LMC-ATL][LMC-MYA]

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

Software Architecture (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Application Programming Interface (API)**
  - **Uses Processing Graph Method (PGM) developed by NRL**
  - **PGM serves as a data flow graph API for signal processing algorithm descriptions**
  - **PGM serves as a command program API for**
    - **Data flow graph execution control (starting, stopping) and monitoring**
    - **Managing I/O devices**
    - **Starting other command programs**
    - **Setting flow graph parameters**
    - **Responding to external inputs**

The Application Programming Interface (API) is a set of functions developed in PGM used to develop data flow applications. These functions serve as a buffer between the application program and the microkernel and need not be changed as the kernel is changed during model year upgrades. The API will be highly transportable from platform to platform.

[LMC-ATL][LMC-MYA]

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

Software Architecture (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Run-time support for three ranges of application requirements**
  - **Static graph mapping to processors with static scheduling (initially)**
  - **Static graph mapping to processors with dynamic scheduling**
  - **Dynamic graph mapping to processors with dynamic scheduling**

Run time support is provided for static and dynamic graph mapping to processors with static or dynamic scheduling.

[LMC-ATL][LMC-MYA]

Software Architecture (Cont.)

- **Operating system requirements**
  - **Service requirements to support RRTS**
    - **Process creation (spawning)**
    - **Protected address space for processes**
    - **Preemptive multitasking**
    - **Support for dynamic priorities**
    - **Round robin time-slicing for equal priority ready tasks**
    - **Mutex and counting semaphores**
    - **Interprocess communication**
    - **Sequential message passing (sockets)**
  - **Support for COTS products with proprietary O/S**
    - **O/S meets service requirements for RASSP**
    - **O/S provides open interface on which the RRTS and associated API can be ported**

 124

The operating systems requirements for the MYA are presented above. It must support the RASSP run-time system (RRTS) and support COTS products with proprietary operating systems.

[LMC-ATL][LMC-MYA]

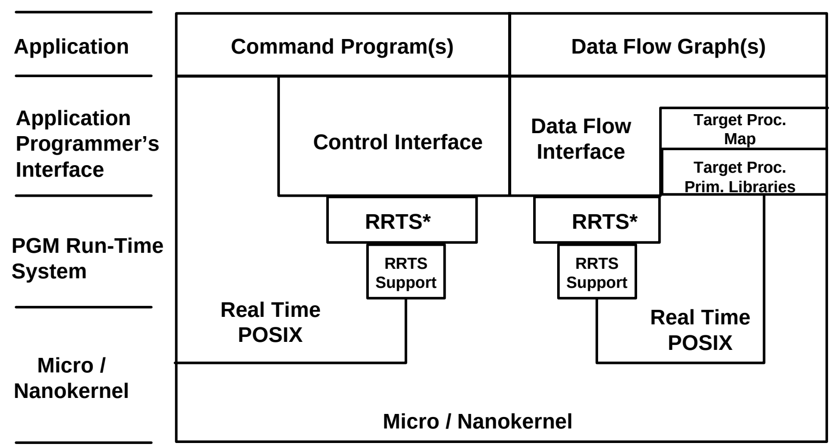
Software Architecture (Cont.)

- **Real-time microkernel/operating system candidates**
  - **Large commercial offering**
  - **Not all suited for high performance embedded signal processing**
  - **Current promising candidates**
    - **SPOX - Spectron Microsystems Inc.**
    - **PSOS+/UNISON - Multiprocessor Toolsmiths**
    - **Real Time Executive for Military Systems (RTEMS) - Developed by On-Line Applications Research Corporation under contract to the US Army Missile Command**
    - **Real-Time MACH - Open Software Foundation (OSF)**

Various real-time microkernels can be used for the operating system. They must be suited for high performance embedded signal processing and a few candidates are listed above.

[LMC-ATL][LMC-MYA]

Software Architecture Diagram



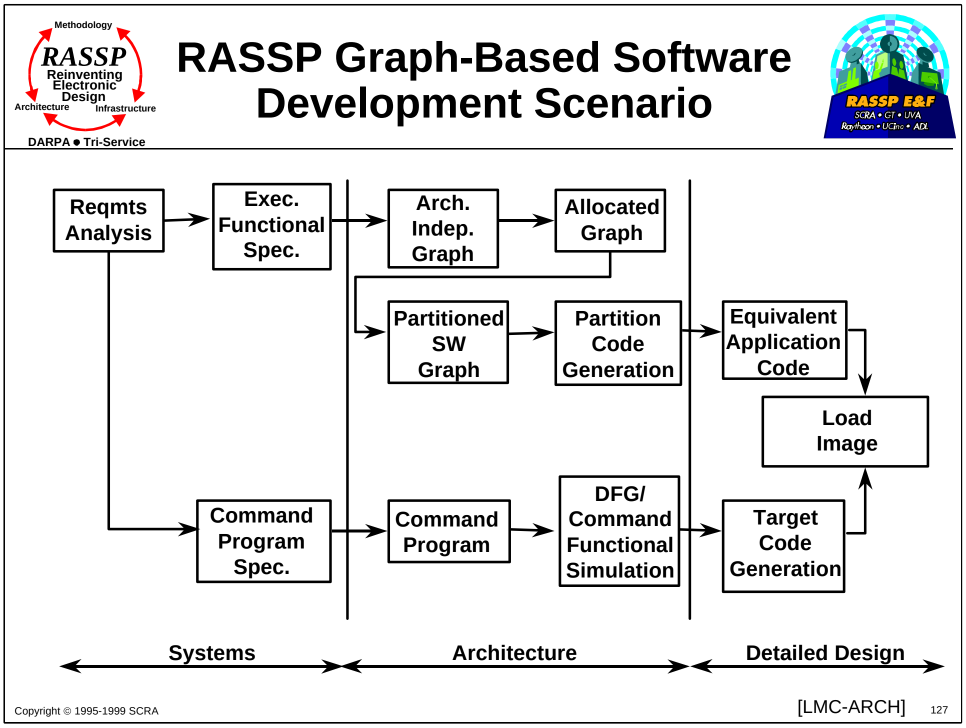
*RRTS: RASSP Run-Time Support

 [LMC-ARCH] 126

The software supports the Model Year Architecture (MYA) concept by providing a common Application Programming Interface (API) to the underlying real-time operating system services. This allows a new hardware platform with a new microkernel to change for each model year while maintaining the API. Support for the API is through the RASSP Run-Time System (RRTS), which provides the services required for the control and execution of multiple graphs on a multi-processor system. The RRTS and its support for the API forms the essential component of software encapsulation for a processor object.

The application layer is divided into two parts. The first part is the command program, which provides response to external control inputs, starting and stopping data flow graphs, managing I/O devices, monitoring flow graph execution and performance, starting other command programs and setting flow graph parameters. The control interface provides services that implement these operations.

The second part of the application layer is the data flow graphs (DFGs) implemented using a data flow language. Services provided by the DFG interface are largely invisible to the developer and include managing graph queues, interprocessor communication and scheduling. The constructed flow graphs will be converted to HOLs such as C or Ada via autocode generation and will contain calls to a standard set of domain primitives.



Software development cannot be discussed without its relationship to the architecture. The software portion of architectural objects is handled by the process shown above.

This process depicts the progression of software generation from the requirements to the load image, with emphasis on the graph objects involved and the general RASSP process in which they occur.

Architecture definition involves the creation and refinement of the DFGs that drive both the architecture design and the SW generation for the signal processor. The DFGs of the signal processor are developed, and the nodes are allocated to either hardware or software. Automated generation of the software partitions is performed to provide executable threads that are to be run on the DSPs. These autocoded partitions are combined into an application graph which is functionally equivalent to the original.

The final step in the SW development, which is the production of the load image, occurs during detailed design. The load image generation is an automatic build process that is driven by the autocode generation results. The inputs to the process include the architectural description, the detailed DFGs describing the processing, the partitioning and the mapping information, the autocode results and the command program.

The process is controlled by a software build management function which extracts the necessary information from the library and manages the construction of all the downloadable code.

Detailed Section Outline

- Fully Functional Modeling
  - Description
  - Case Study
- Model Year Architecture
- **Reuse**

We address the final aspect of RASSP methodology, which includes "reuse" at multiple levels of abstraction. This includes reuse of past designs, past software and hardware libraries, reuse of modeling and simulation environments, etc.

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

How Re-use Fits into RASSP

- **The model year approach designs and builds a system iteratively**
- **In order for this to work, previous versions of the design must have been made in a way that enables them to be reused with minimum rework. This would include elements like system software and processor architecture**
- **Each subsequent "model year" should take less time than the previous because of the reuse philosophy**
- **Definition of a Re-Use element**
  - **Any design element that is complete and consistent to the extent that it can provide value in a new design**

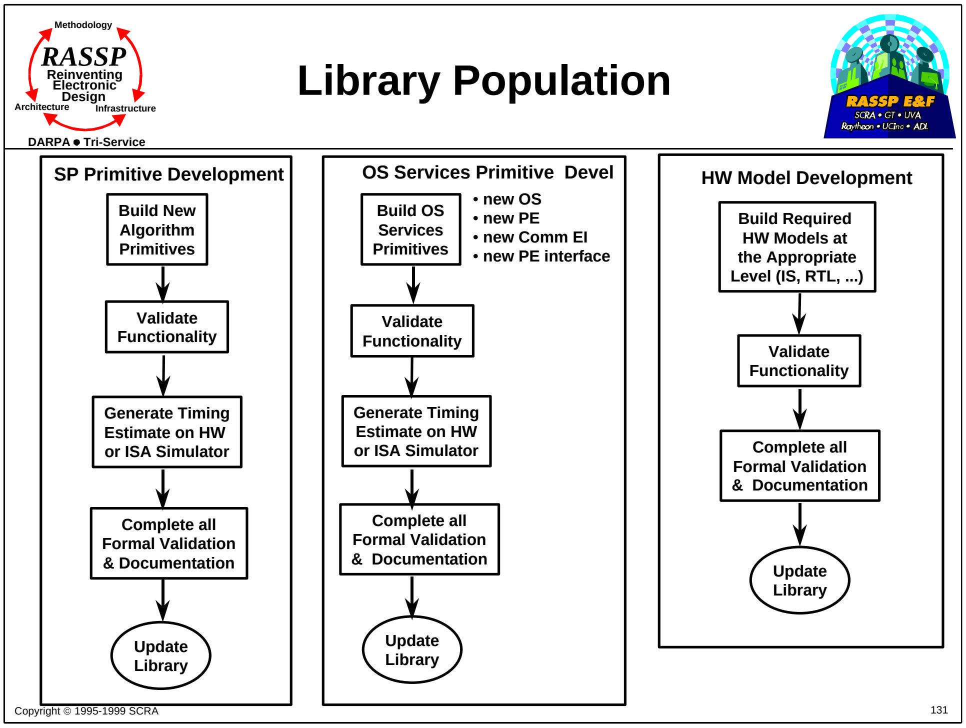
Examples:

- A bus controller card design, including related schematics, VHDL models, timing, software, test data, etc.
- An ASIC design, including related artwork and schematics, VHDL model, test data, software, etc.
- A released catalog entry for an individual component, including component properties, schematic symbol, etc.

Contents of Reuse Library

Software Reuse Library	Hardware Reuse Library
• SW Performance Models	• Performance models
• Application code / code fragments	• Behavioral models
• OS Kernel(s) / OS services	• RTL models
• Application DFGs	• DFG partitions and mappings
• Control/support software	• Architecture configurations
• Test data	• Test plans and test sets
• Documentation elements	• Documentation elements

The contents of the hardware and software component reuse library has models and data at various levels as shown in the chart above. These models support concurrent codesign throughout the selection and verification process. The reuse library drives both the architecture synthesis and the software synthesis processes in an integrated fashion.



When any of the required support components are not present in the library, one must define (at least) a prototype element. Adding any component function to the library "library population".

The library set must be extended over time to accommodate new technology and applications.

There are three library population software development activities that must be supported:

1) building a signal processing primitive library element,

2) building an operating system primitive library element, and

3) developing hardware models.

For operating system services primitive development, there are four instances where one must generate or modify operating system services:

1) new operating system

2) new processing element

3) new communication element

4) new processor interface

RASSP Methodology Overview Outline

- Overview
- Methodology
  - Overview
  - Virtual Prototyping
  - Model Year Architecture
  - Reuse
- **Results to Date**
- Summary

We present some of the several results from the RASSP program. The reader is requested to refer to the proceedings of the two RASSP conferences for detailed discussions of these results.

Results to Date

Significant <u>use</u> and success of RASSP spans a number of projects:

- **UAV / ATDS SAR** Signal Processor Development
- **AIRMS IRST** Development and Insertion
- **F-14D IRST** (AN/AAS-42 WRA-2) Technology Upgrade
- **SH-60 ALFS / LAMPS** (AN/UYS2a FPCAP) Technology Upgrade
- **ACOMMS** SONAR Buoy Upgrade
- **NSA "Kindling"** Model Year Studies
- **JAST** Integrated Sensor System (ISS) Development
- **F-15C** (APG-63U) Multi-mode RADAR Reliability Upgrade and Insertion

Projects have funding or support from the SPO / PMA / PMO / PME of the platform and module with plans for flight test of the successful systems produced. Projects will yield first pass, form / fit / (enhanced) function prototype systems.

Sites testing or using RASSP: Alliant Tech, ARL, Hughes, Johnson Space Center, LM Baltimore Labs, LM Comm, LM Electronics, LM Orlando, Motorola, NSA, Sanders, TRW, Woods Hole, Wright Laboratories

RASSP has now spurred several developments in the commercial arena as well, with terms like behavioral models, virtual prototyping, design-with-resuse becoming standard terminology within the industry.

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

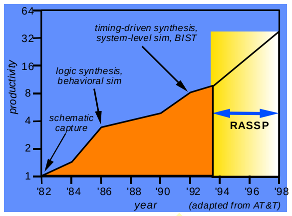
RASSP Methodology Overview Outline

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Overview
- Methodology
  - Overview
  - Virtual Prototyping
  - Model Year Architecture
  - Reuse
- Results to Date
- **Summary**

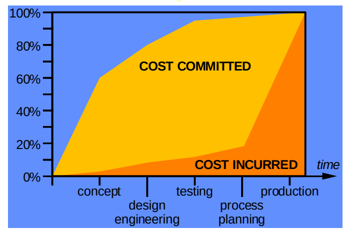
Summary: Addressing System Level Design Enables Achievement of 4x

Productivity doubles every three years (and lags technology performance improvement!)



Virtual Prototyping and Model Year Architectures allow the discovery of problems early in the design cycle

RASSP 4x will come from applying more rigorous tools and methodologies earlier in the design process



Production costs are committed early in the design cycle; long before the actual cost

135

RASSP had advanced system-level design to a new arena above and beyond traditional practice. We describe the many steps of the RASSP process in other E&F modules.

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

Summary

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **RASSP Seeks a 4x Improvement in Design Time, Quality and Life-Cycle Cost for Embedded DSPs**
- **RASSP is focused at formalizing the early stages of design, in assisting with design re-use, and with enabling model year upgrades through the effective use of virtual prototyping and HW/SW Codesign**
- **Demonstration of significant cost and cycle time savings using the RASSP methodology and tools has been shown**
- **Process change and education is being developed, demonstrated, and proliferated**

 136

In the past, the commercial Electronic Design Automation (EDA) and the academic/industrial research communities have been aware of the requirement for an intensive effort to study the digital system design process in its entirety; however, resource needs, fuzzy objectives, and short-time horizon have handicapped progress. Currently, the Rapid Prototyping of Application Specific Signal Processors (RASSP) program is overcoming these handicaps and is developing a number of new technologies that will lead to shorter prototyping times, improved productivity quality and reduced life cycle costs.

Successfully transferring the technology being developed by the RASSP program to industry and academia is a critical component of the overall RASSP effort. To accomplish this goal, a novel, ground breaking RASSP Education & Facilitation (RASSP E&F) program was explicitly funded and a team tasked with leading the RASSP efforts to transfer technology from the RASSP program to the university and industrial communities.

Exercises

- Describe the problems with the current system design practice?
- What is cost modeling and determine how it can provide savings on the design and life cycle costs.
- What is the difference between performance modeling and fully behavioral modeling?
- In the table comparing the various detailed integration and test methodologies, try to justify the various metrics and their costs provided.
- Why would upgrades be simpler with a model year architecture? What about legacy systems?
- Is VHDL necessary to implement the RASSP methodology?

References

[DeBardelaben 97] J. DeBardelaben, V. Madisetti, and A. Gadient, "On Incorporating Cost Modeling in Embedded Systems Design," *IEEE Design & Test of Computers*, Vol. 13, No. 3, July 1997.

[Dung 96] L-R Dung, V. Madisetti, "Conceptual Prototyping of Scalable Embedded DSP Systems," IEEE Design & Test of Computers , Vol 13, No 3., Fall 1996.

[Egolf 96] T.Egolf, et al, "VHDL-Based Rapid System Prototyping, "*Journal of VLSI Signal Processing*, Vol. 14, Issue 2, 1996.

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

[LMC-ATL] Caracciolo G., Pridmore J., "*Architectures for Rapid Prototyping of Embedded Signal Processors"* , The RASSP Digest, Vol. 2, No. 1, 1st Quarter 1995.

[LMC-MYA] Caracciolo G., *" Second RASSP Model Year Architecture Working Group Meeting"* , Martin Marietta Laboratories, March 15, 1995.

[LMC-ARCH] Shamming B., *"RASSP Methodology Working Group Meeting Architecture Process"* , Martin Marietta Laboratories, March 16, 1995.

[LMC-Meth] *"RASSP Methodology Version 1.0"* , Martin Marietta Laboratories, December, 1994.

[Hein98] Hein, et al, "RASSP VHDL Modeling Terminology and Taxonomy," Version3.0, July 29, 1998.

[Madisetti 94] V. Madisetti,"Vive La Difference," The RASSP Digest, Vol 1, 4th Quarter 1994.

[Madisetti 95A] V.Madisetti, T. Egolf, "Virtual Prototyping of Embedded DSP Systems," *IEEE Micro*, pp. 9-21, Fall 1995.

[Madisetti 95B] V. Madisetti and J. DeBardelaben, "A RASSP Approach to HW/SW Codesign, The RASSP Digest, Vol. 2, 4th Quarter, 1995.

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

References

[Madisetti 95C]V.Madisetti, T. Egolf, "Virtual Prototyping of Embedded DSP Systems," IEEE Micro, Fall 1995, pp. 9-21

[Madisetti 96]V.K. Madisetti, "Rapid Digital System Prototyping: Current Practice, Future Challenges, IEEE Design & Test of Computers, Vol. 13, No. 3, Fall 1996.

[Malley 96] J. Malley, "RASSP: Changing the Paradigm of Electronic-System Design," IEEE Design & Test of Computers, Vol. 13, No. 3, Fall 1996.

[Richards 97] M.A. Richards, A. Gadient, G. Frank, R. Harr, "The RASSP Program: Origin, Concepts, and Status," Editorial in Journal of VLSI Signal Processing Systems, Volume 15, No. 1, 2, February 1997, pp 7-28.

[Richards 97] M.A. Richards, A. Gadient, G. Frank, eds., Rapid Prototyping of Application Specific Processors, Kluwer Academic Publishers, Norwell, MA, 1997.

[Saultz 97] J. Saultz, "Rapid Prototyping of Application-Specific Signal Processors (RASSP): Progress Report," Journal of VLSI Signal Processing Systems, Volume 15, No. 1, 2, February 1997, pp. 29-48.