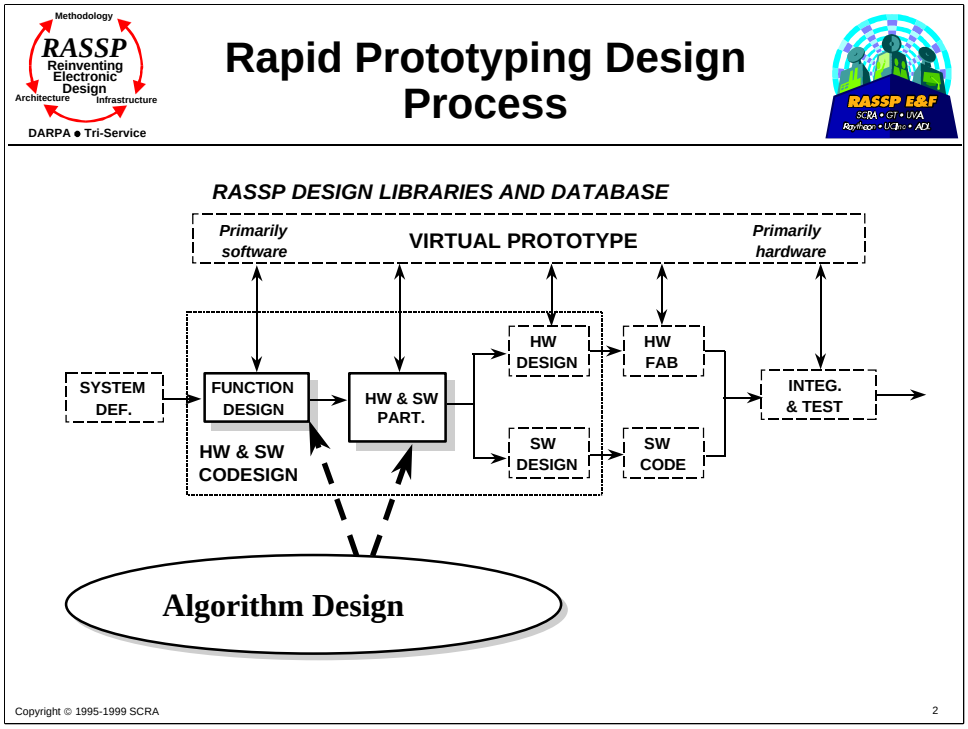
DSP Algorithm Design

RASSP Education & Facilitation Program

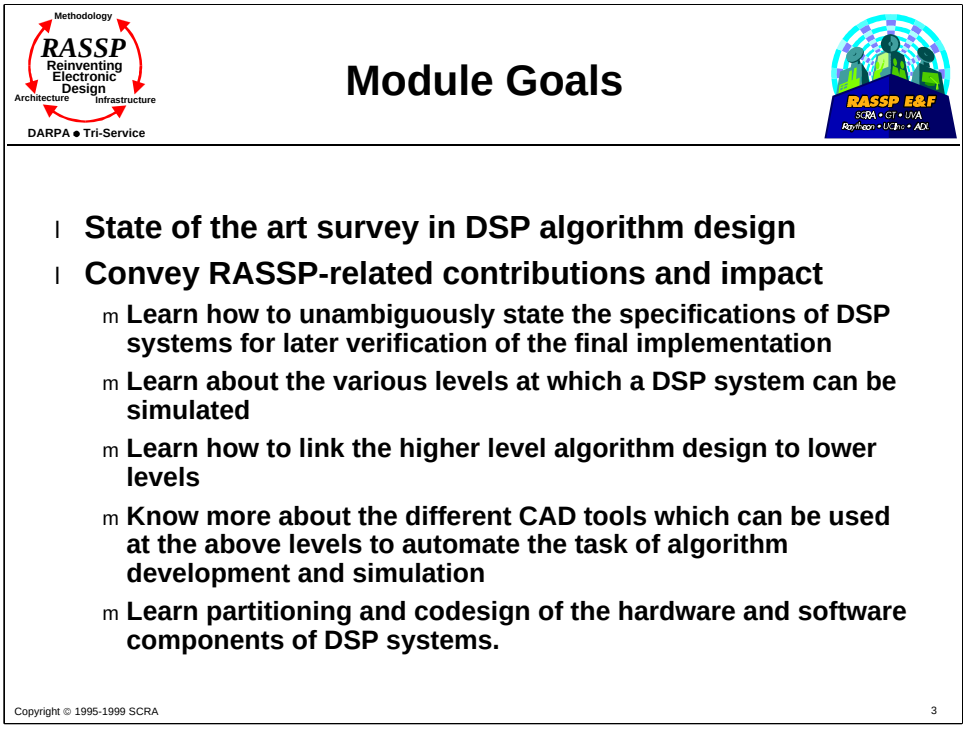
Module 23

Version 3.00

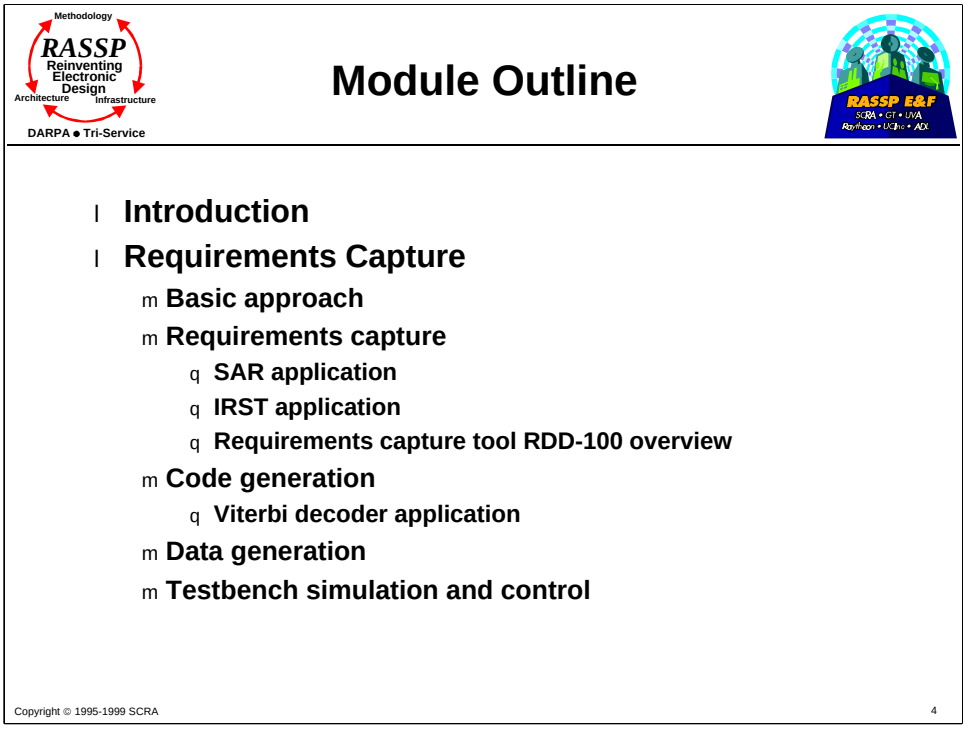


Algorithm design begins with the analysis of the design specification, and guides the algorithm developer to the point of HW/SW partitioning. This includes the domain of functional design.

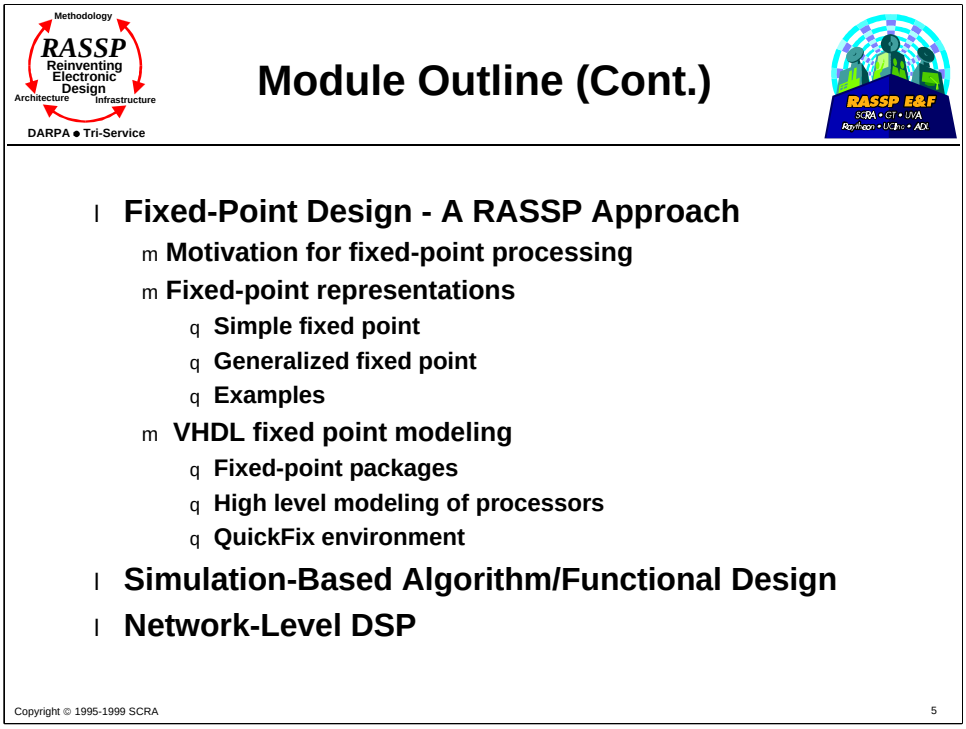
Various environments and methodologies will be discussed throughout this presentation to allow the algorithm designer the ability to extract and analyze specification, develop algorithms for these applications, and finally do initial trade-offs on architecture selection through HW/SW partitioning.



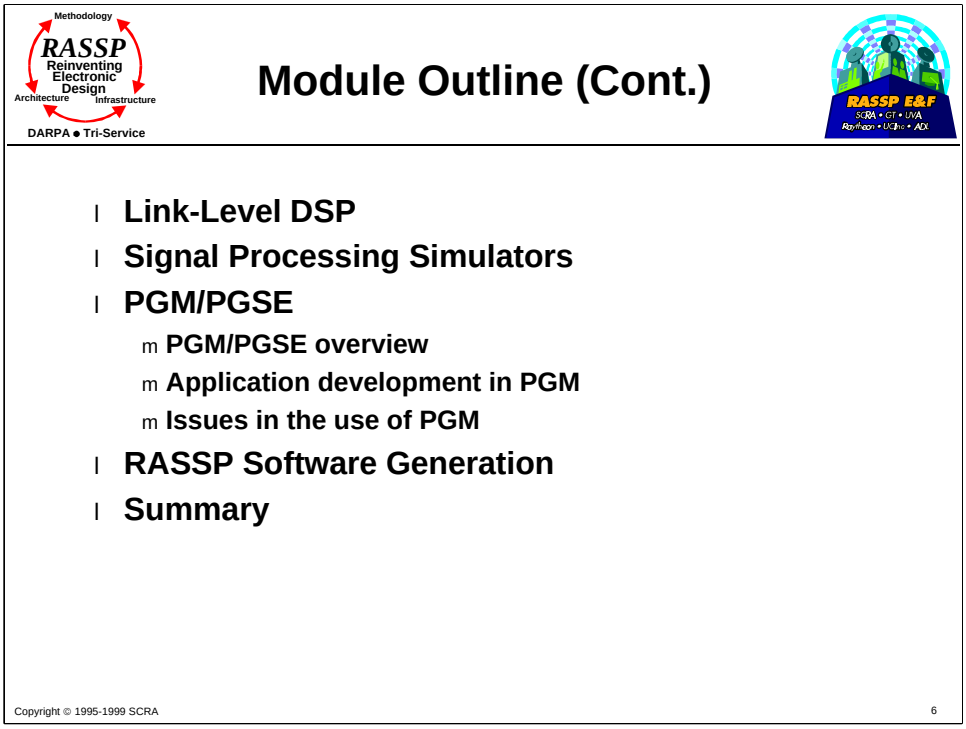
We will describe how DSP algorithm and application requirements and specifications are captured in an executable form.



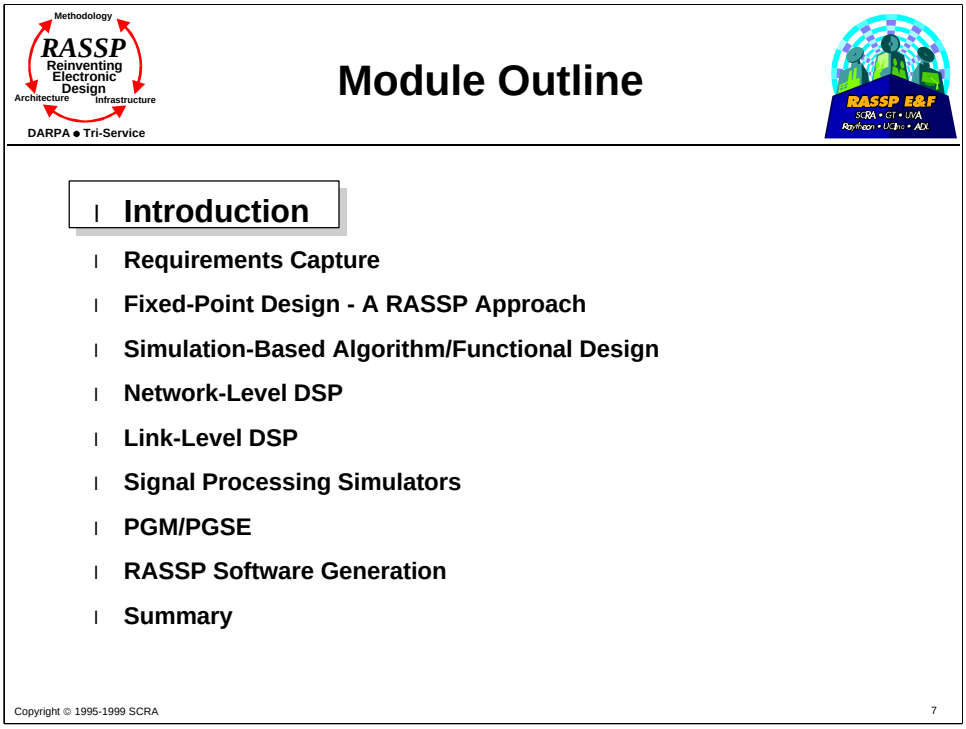
Requirements are generated by the customer specifying the input output properties of the intended application. Requirements specify what the system should do, as opposed to how it should do it.



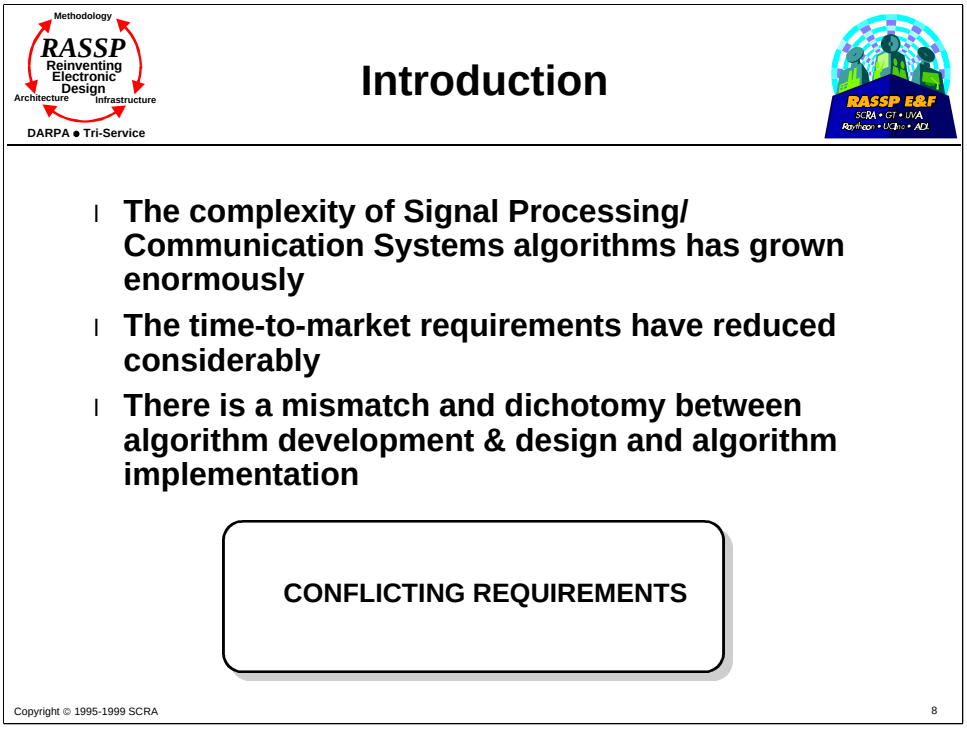
Continuation of table of contents



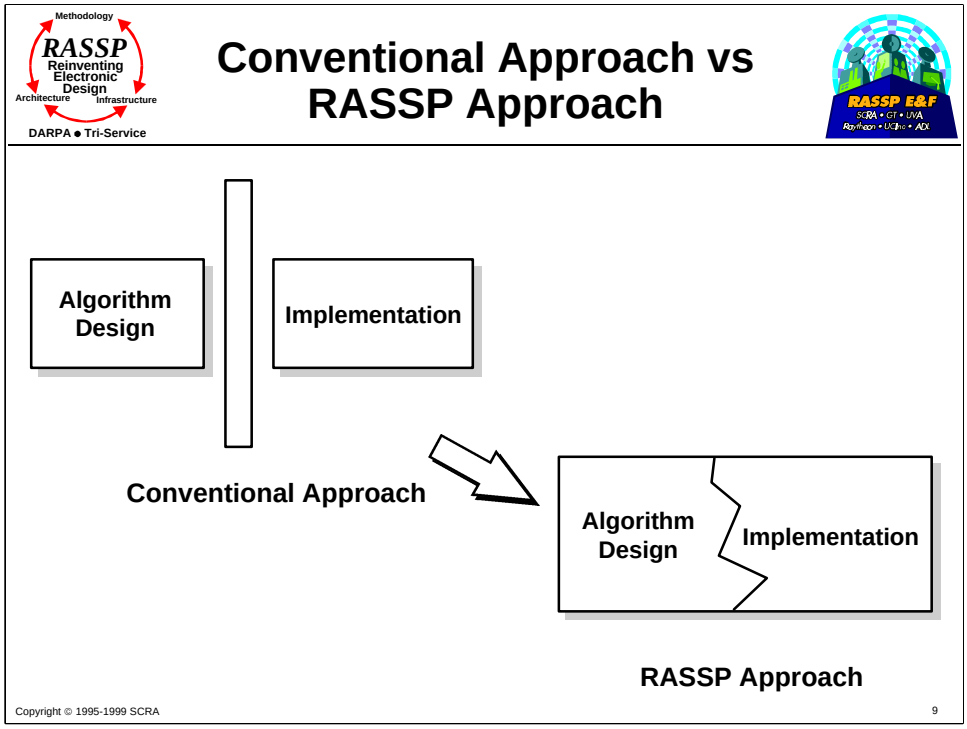
Continuation of table of contents



We now describe how requirements can be captured in an executable form.

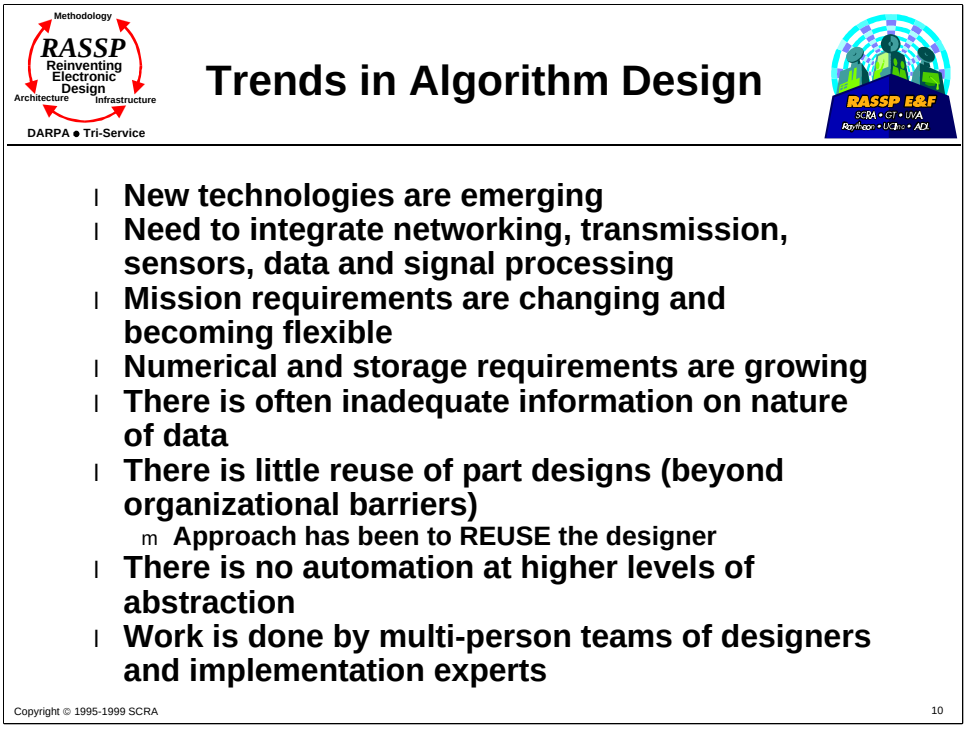


This motivates the need for a structured approach to algorithm design. The constraints on the designer require a rapid methodology to obtain a final HW/SW solution to a specified application.



In the conventional approach for designing a DSP system the higher level algorithm development and the lower level hardware implementation are independent of each other.

In the RASSP approach both the higher and lower levels of the design are dealt with simultaneously.

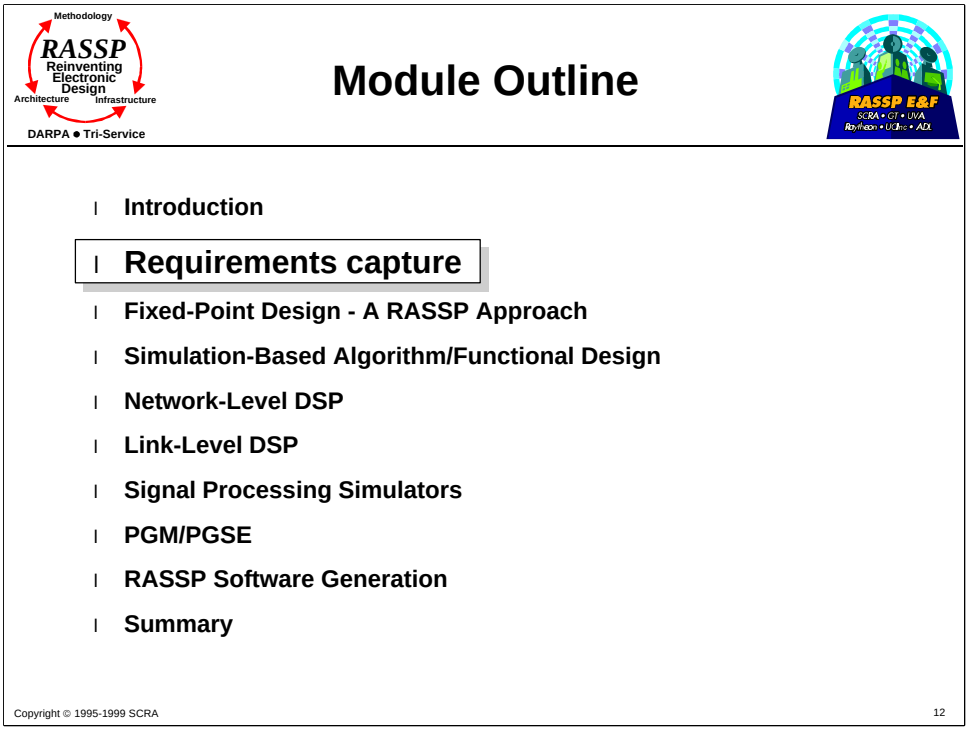


This slide describes the changing nature of communications/DSP system design given the changing scenarios of use, combined with rising design costs.

Requirements of RASSP Algorithm Development Environment

- **Model, analyze, design, and test large systems (at multiple levels of abstraction)**
- **Allow rapid exploration of a variety of reuse-based sources**
- **Link the algorithm design environment to lower levels of abstraction**
- **Continue HW/SW modeling and use real data to produce effective designs**
- **Allow efficient multi-person team collaboration (product, design, and management teams)**

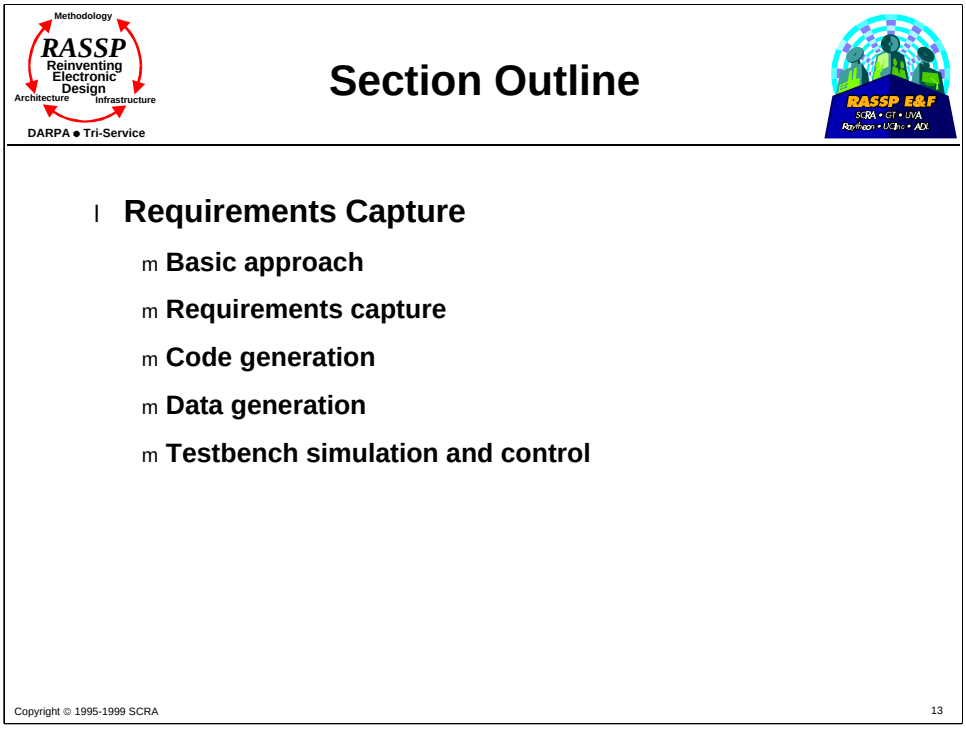
This slide outlines the wish list of features required of a RASSP-like environment.



Given the previous wish-list, we would now like to start at the top and:

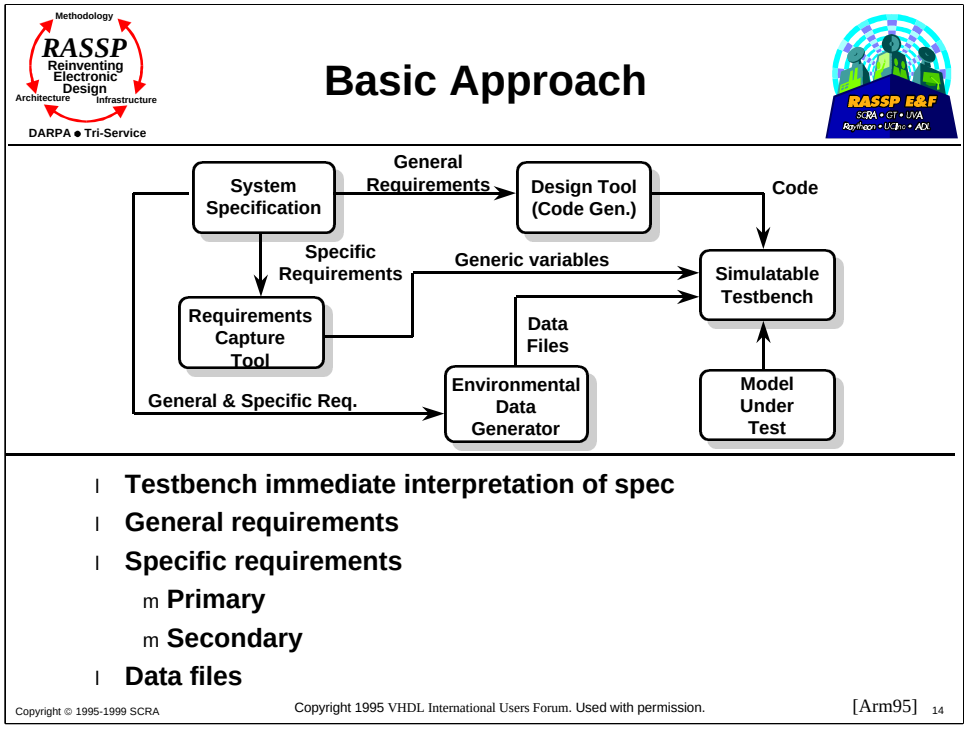
- Look at capturing the requirements of a given specification
- Show how these requirements can flow down the various stages of the design process.

We start by looking at the testbench, which is the immediate interpretation of the specification that our design must come up against through the various levels of design abstraction.



This section will discuss the generation of high-level testbenches in VHDL for studying the requirements of an algorithm.

The above topics on testbench generation will be discussed, and an environment will be presented to perform this methodology.



One of the goals of the DOD DID for electronic parts is to use testbench as an immediate interpretation of the specification.

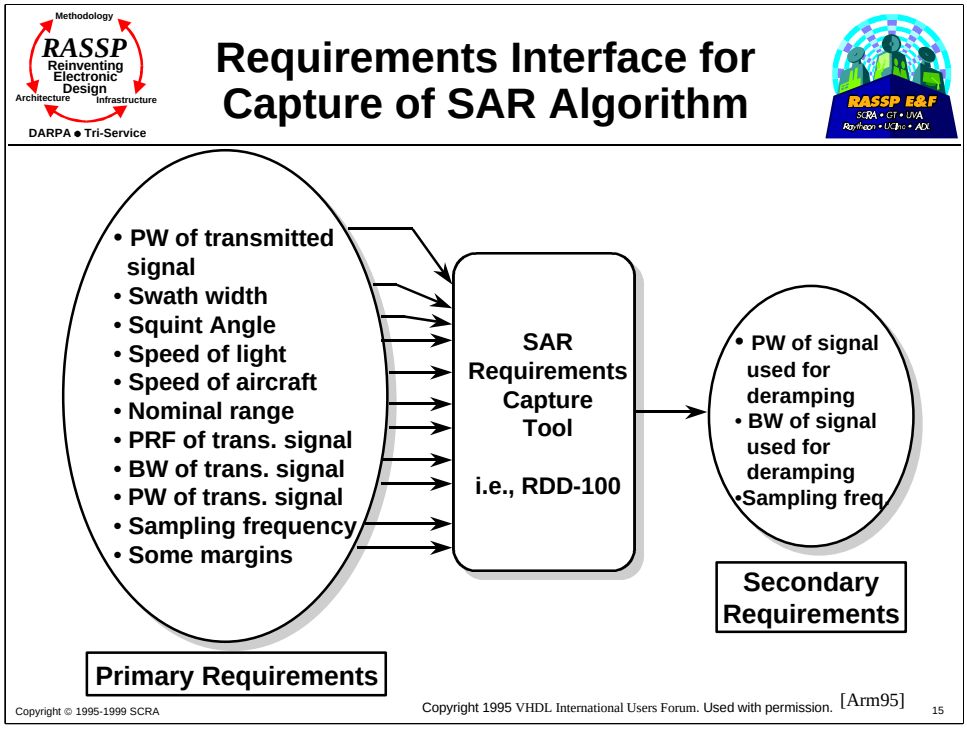
The above figure is a high-level testbench generation system to implement this strategy.

The system specification contains both general and specific requirements which together specify the system.

General requirements specify the class of system being modeled.

Specific requirements select a particular member of that class of systems. There are primary and secondary specific requirements, and these will be mentioned in more detail later.

Data files can be generated from either general or specific requirements. These files are read by the testbench during execution using file I/O.



This diagram illustrates the requirements interface for the SAR algorithm. The requirements capture tool can be the RDD-100, for example.

This diagram also shows the primary and secondary requirements for the SAR algorithm.

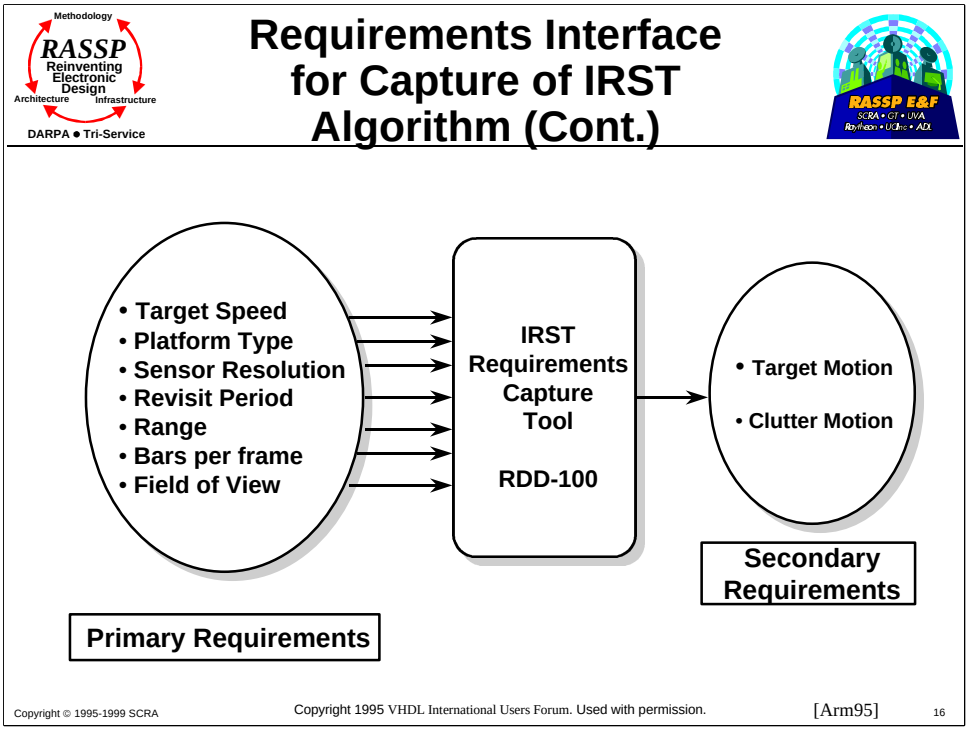
The secondary requirements are derived from the primary requirements through use of the mathematical modeling capabilities of the capture tool.

Primary requirements

- Squint angle
- Carrier frequency
- Swath width
- Nominal range to center of swath
- Pulse repetition frequency of transmitted signal
- BW of transmitted signal.

Secondary requirements

- Pulse width of signal used to do deramping
  - = PW of trans. signal + swath width/speed of light
- Bandwidth of signal used to do deramping
  - = rate of change of freq.* PW of signal
- Sampling frequency
  - sampling frequency > 2 * carrier frequency
- PW of transmitted signal
- Speed of aircraft
- Sampling frequency for the resampling process.



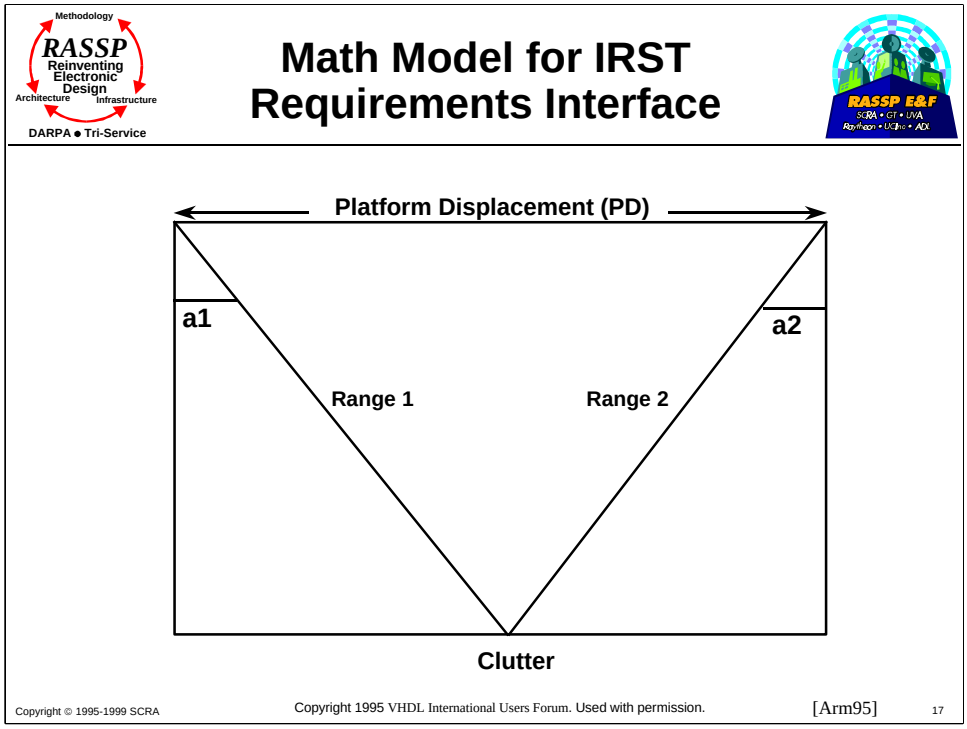
This is a similar diagram used for the IRST algorithm. The primary and secondary requirements are listed. Their derivation is on the next slide's note page.

Primary requirements

- Target speed in Machs
- Platform type (three types VF-X, VF, VP)
- Sensor Resolution (High or Low)
- Revisit Period (frame update rate, fed as generic to testbench)
- Range of clutter and target from platform

Secondary requirements

- Target motion (pixels per frame)
- Clutter motion (pixels per frame)



This slide shows a plot of the math model for the IRST requirements. The derivations for each of the parameters are listed below.

Total angular disp. = a1 + a2

Sin(a1) = (.5*PD)/range 1

a1 + a2 = 2*asin(PD/(2*range 1))

PD = (a1 + a2)/PAFOV

Sensor resolution factor =

- 100/1000000 if sensor resolution is high
- 250/1000000 if sensor resolution is low

Platform velocity =

- 448 m/s if platform type = VF-X
- 256 m/s if platform type = VF
- 192 m/s if platform type = VP

Clutter range = 1609.344 * range

Platform Disp. = Platform velocity * Revisit period

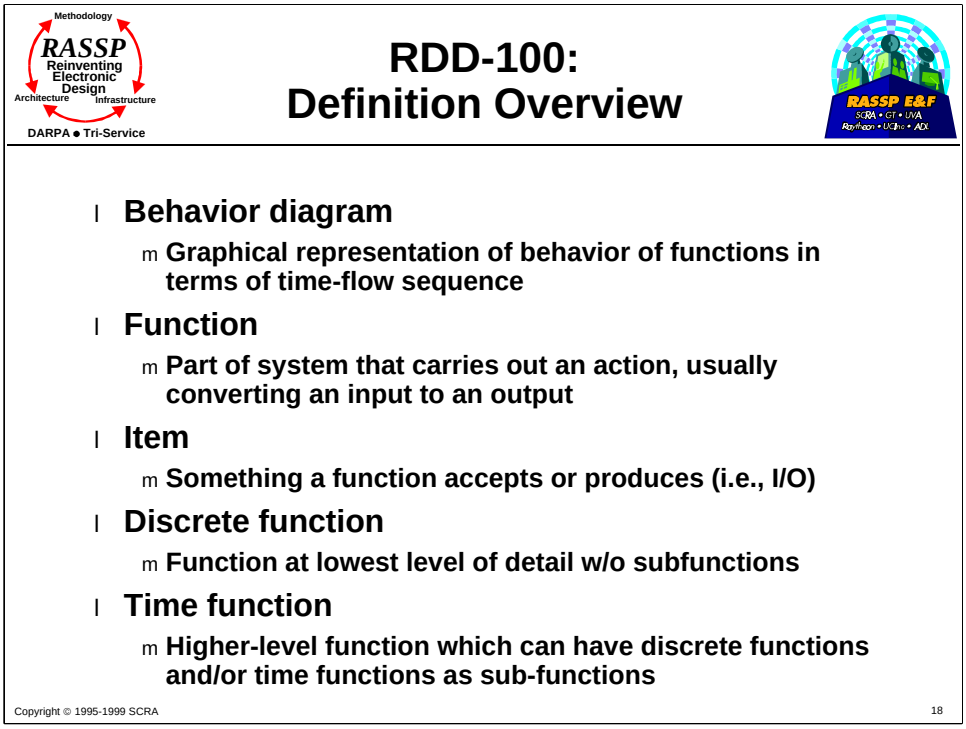
Clutter motion = (2*asin(platform disp/(2*clutter period))/sensor res. factor

Target range = 1609.344*range

Target velocity = Mach to m/s * target range

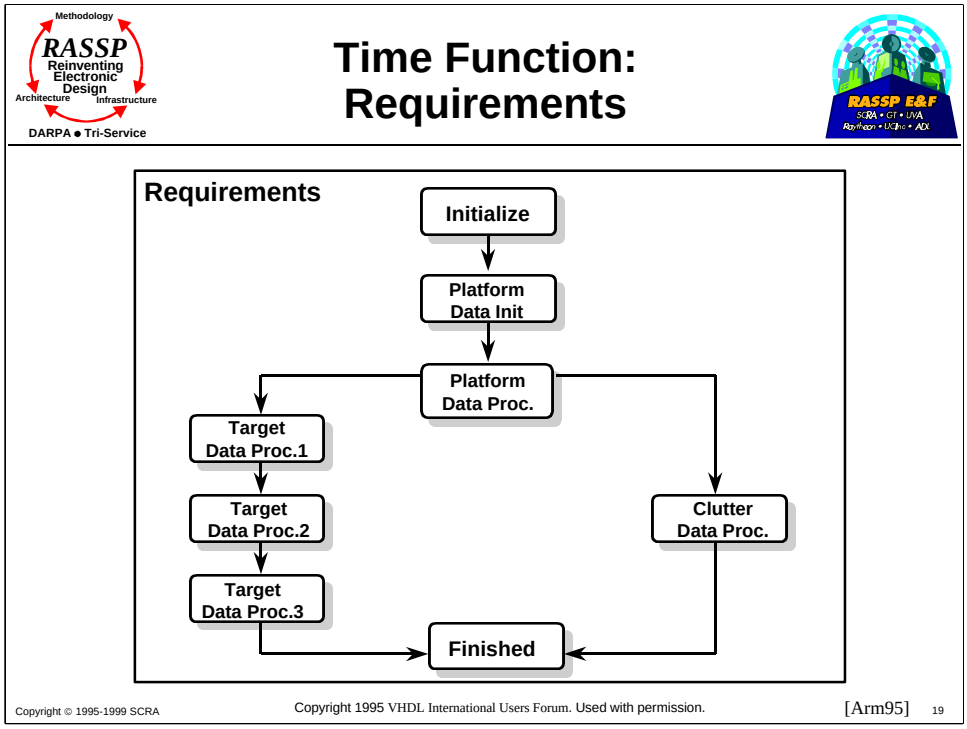
Target disp. = Target velocity * revisit period

Target motion = (2*asin(target disp./(2*target range))/sensor res. factor



[Arm95]

The above chart gives some general terms and constituent components associated with the RDD-100 environment. The following slide uses some of the definitions to describe a high-level time function developed for the IRST system.

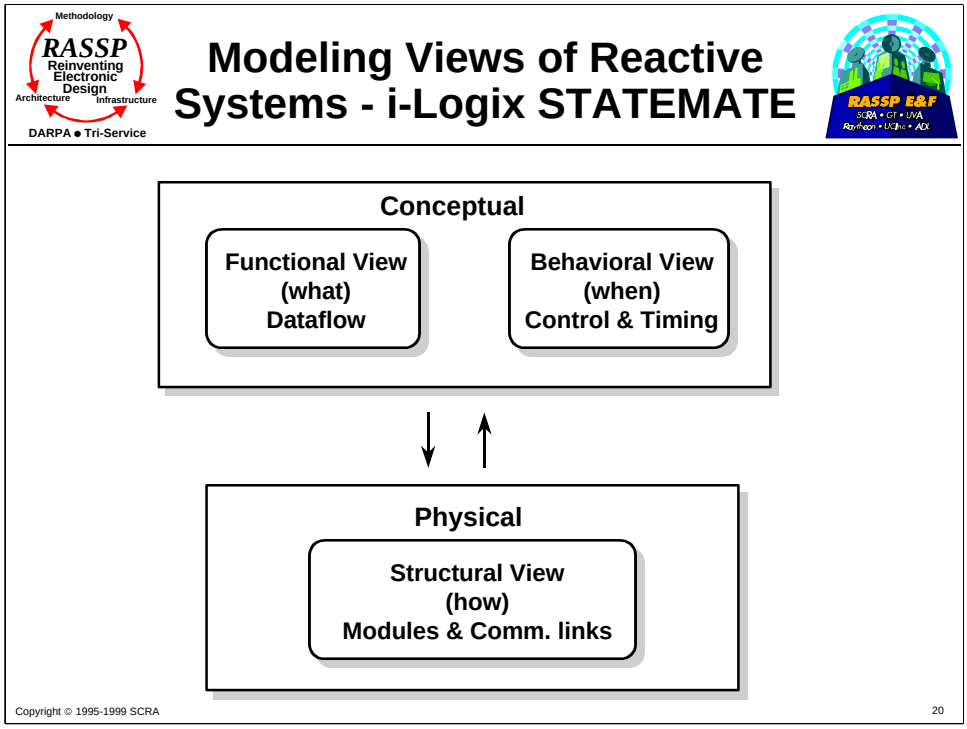


The highest-level time function developed was Requirements. It was decomposed into discrete functions: Initialize, Platform Data Init., Clutter Data Proc., Target Data Proc 1, Target Data Proc 2, and Target Data Proc 3. The inputs, outputs, and intermediate values were stored as discrete items.

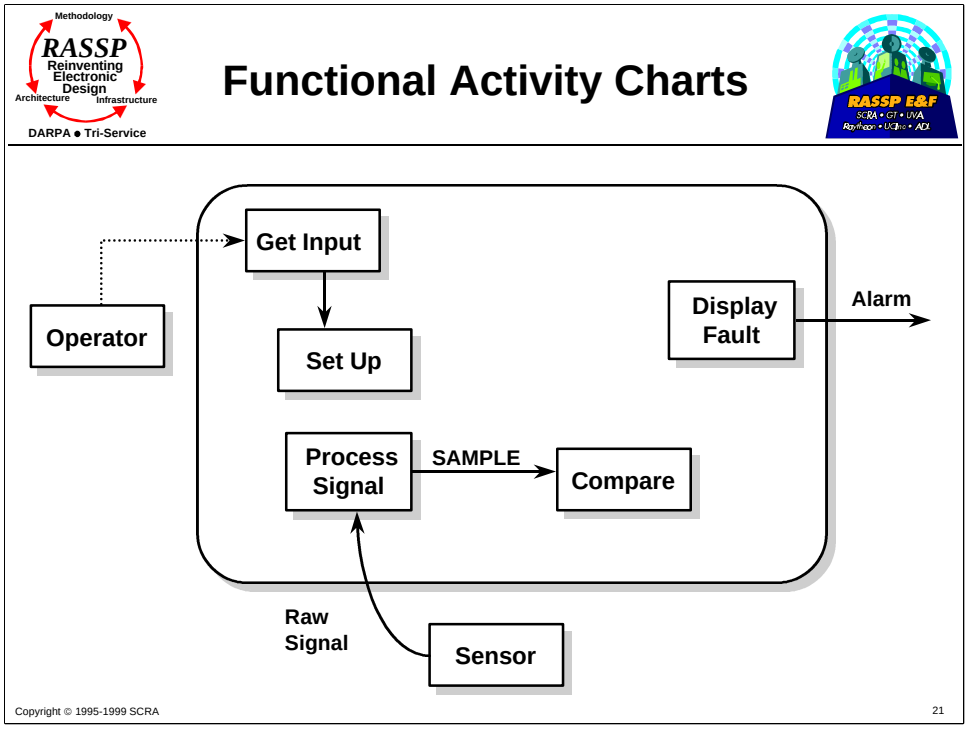
A description of each of discrete function follows:

- Initialize: used to assign values to the primary inputs, target speed, platform type, etc.
- Platform Data Init: used to initialize the conversion factor mach to m/s based on platform type and velocity
- Clutter data proc: used to derive Clutter Range using Range and Clutter Motion from Clutter Range
- Target Data Proc 1: used to calculate Target Range from Range and Target Velocity based on the mach m/s conversion factor and Target Speed in machs
- Target Data Proc 2: used to calculate Target Motion based on the calculated Target Disp, Target Range and Sensor Res. factor.

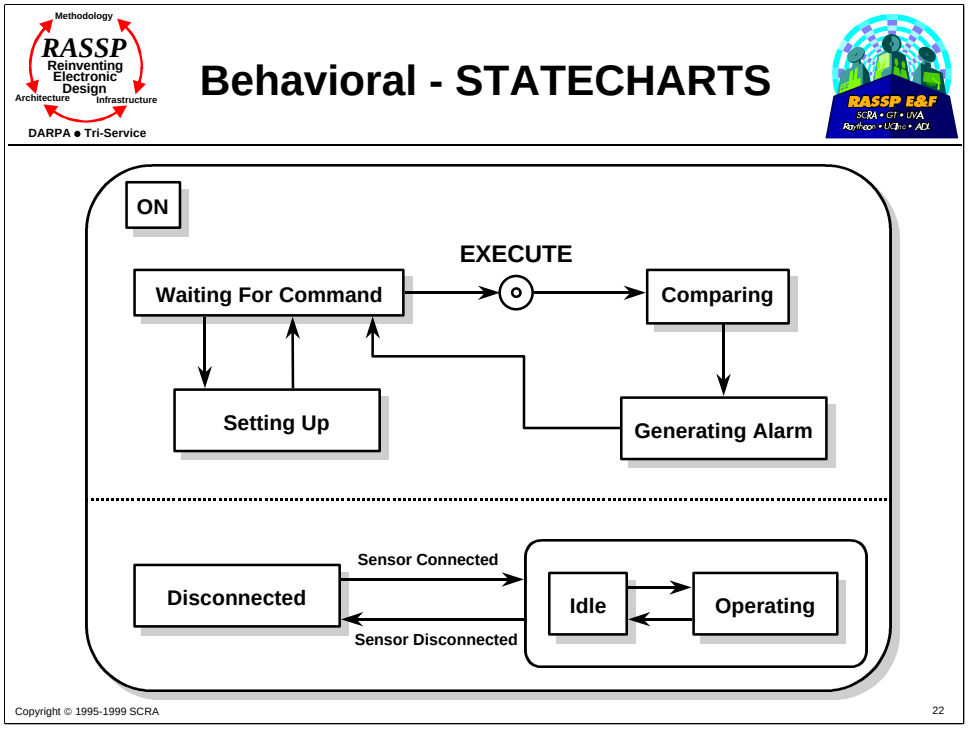
The Requirements time function was verified using the Dynamic Verification Facility of RDD-100. The primary requirements were assigned to their respective items and the secondary requirements were observed.



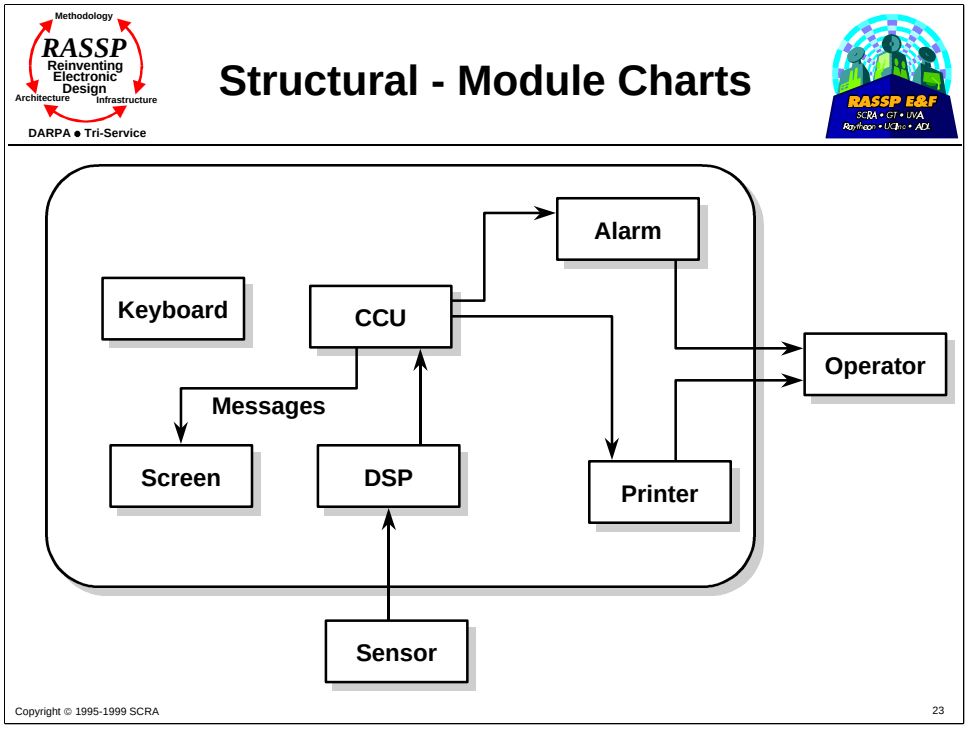
Statemate is a graphical methodology for rapidly specifying reactive systems. The types of characteristics found in a system include both conceptual and physical. Conceptual characteristics include functional views represented as data flow elements and behavioral views for control and timing. The physical characteristics include elements for representing structural views of elements such as modules and communication links. The reader may see more recent developments that greatly extend Statemate capabilities in the Unified Modeling Language (UML) methodology for requirements specification. (Additional details are available at www.rational.com).



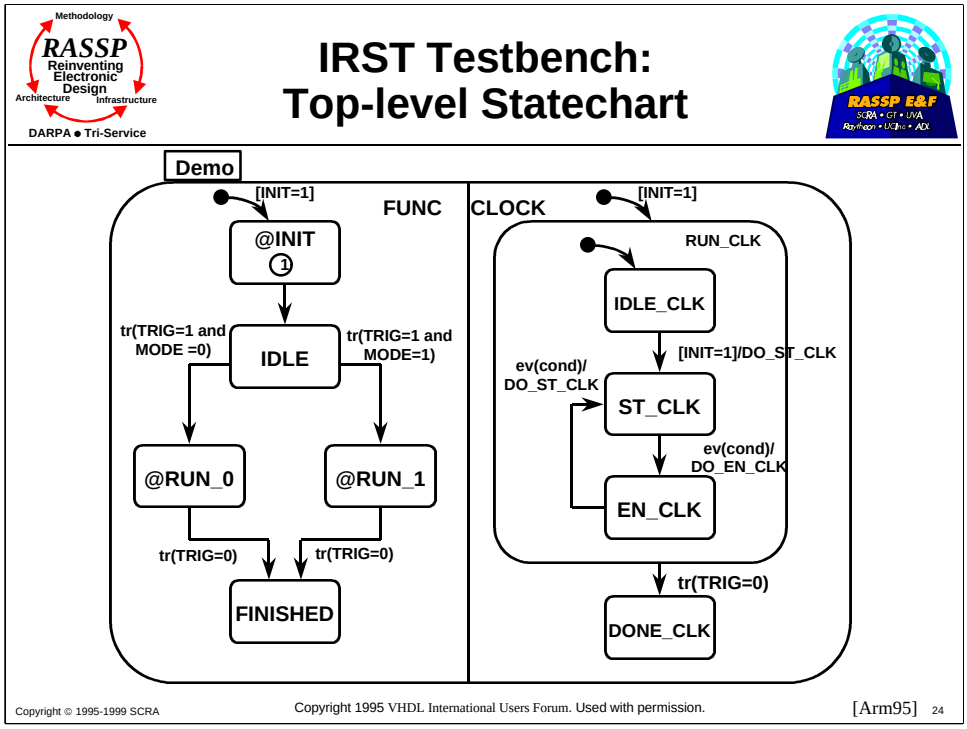
Example of a functional activity chart in i-Logix for describing the list of functions and dataflow.



Example of the timing behavior of an alarm system captured as part of i-Logix. Note: concurrency in activities.



The module list is similar to an equipment list that describes the allocation of hardware.

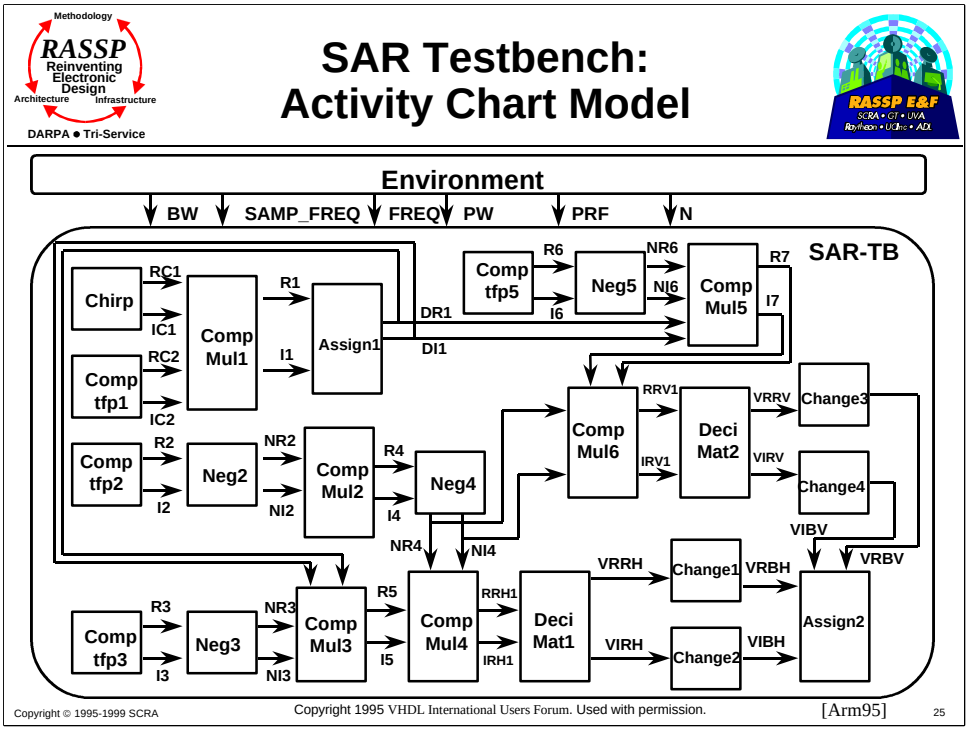


Statecharts are modified versions of state transition diagrams. They are made up of states and transition from one state to another. A label event(cond)/action is placed on each transition arrow describing the condition for transition and action to be taken.

The above figure shows the statechart for the IRST testbench. There are two concurrent states, FUNC and CLOCK. CLOCK is responsible for generating the clock for the output frame generation. The period is an input parameter. FUNC describes the control flow behavior associated with the testbench. INIT and TRIG control the entire flow of the testbench. INIT initializes signal and default transitions and starts the clock. TRIG controls the starting and stopping of frame generation. TRIG high causes frame generation.

Lower-level statecharts exist for INIT, RUN_0, and RUN_1. These perform the actions of the specified mode of operation.

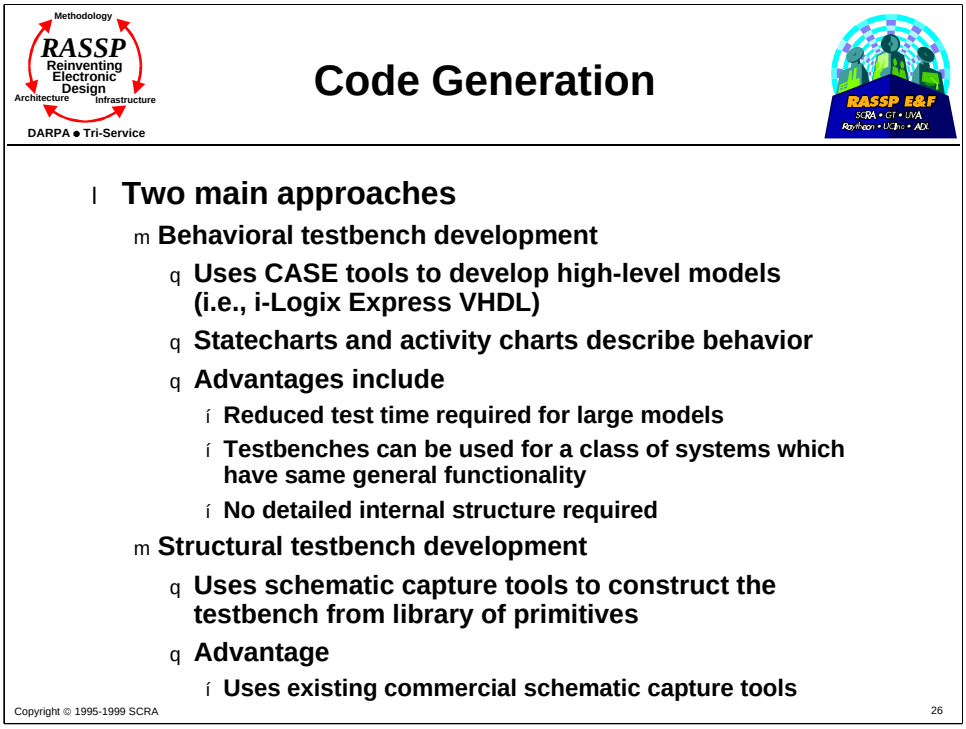
VHDL and Verilog code generation is possible from the statechart tool because there are underlying templates for each of the actions.



Data flow behavior is represented by the use of activity charts as found in Express VHDL and shown above.

The operation of the SAR testbench is described as follows:

- First, the transmitted signal is produced by generating a chirp and then multiplying it with a complex tone.
- The received signal is now simulated by delaying the transmitted signal.
- The two signals are then passed to the two down-converters. This is done by multiplying them with the complex conjugate of the complex tone.
- They are then sent to the deramp section, where correlation is done between the received and transmitted signals.
- The complex conjugate of the down-converted transmitted signal is multiplied with the down-converted received signal.
- The output of the deramping section is fed to a decimation section, where the samples are reduced for analysis.
- The output of decimation is of type real and is converted to bit-vector (40-bit) and assigned to the final output.
- These 40-bit words are used to feed to the SAR processor model as test cases.

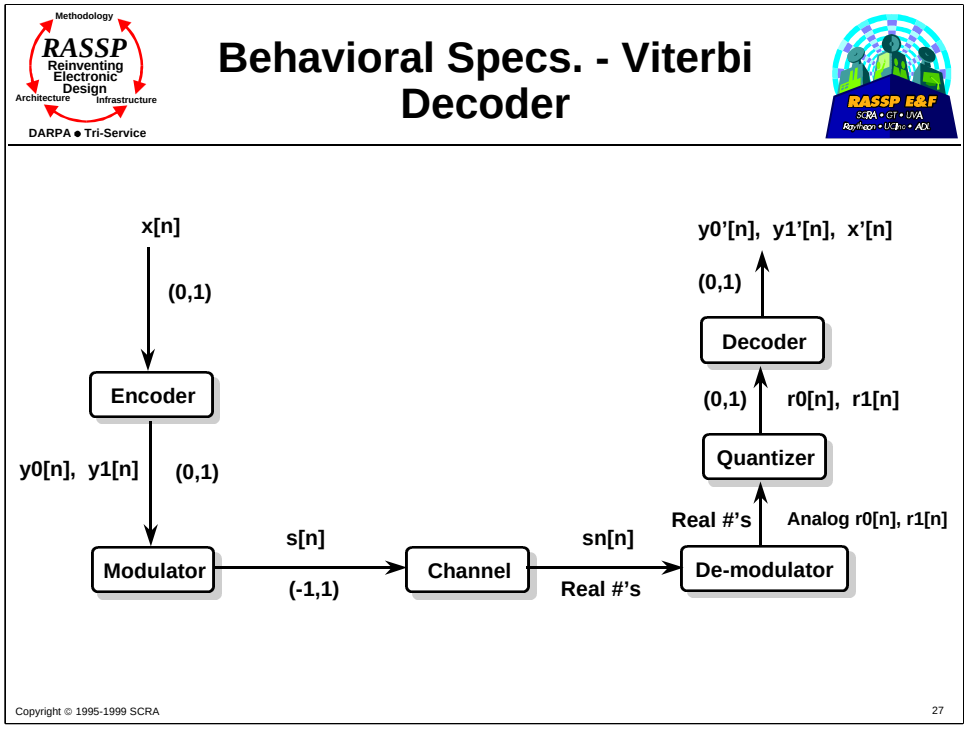


[Arm95]

Code generation can be done from CASE tools such as i-Logix's Express VHDL.

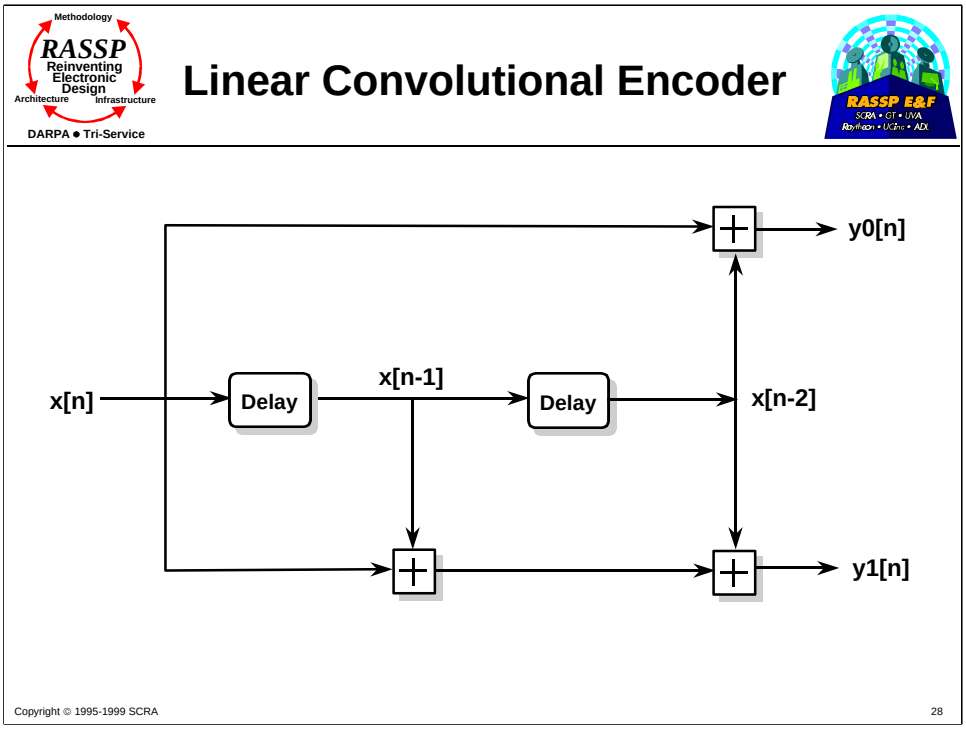
Two approaches to code generation will be discussed in the following slides. These include behavioral testbench development (using CASE tools) and structural testbench development (using schematic capture tools). The advantages of using each approach are listed above.

The CASE tool used for the first approach was i-Logix's Express VHDL which uses Statecharts to describe control flow and activity charts to describe data flow.

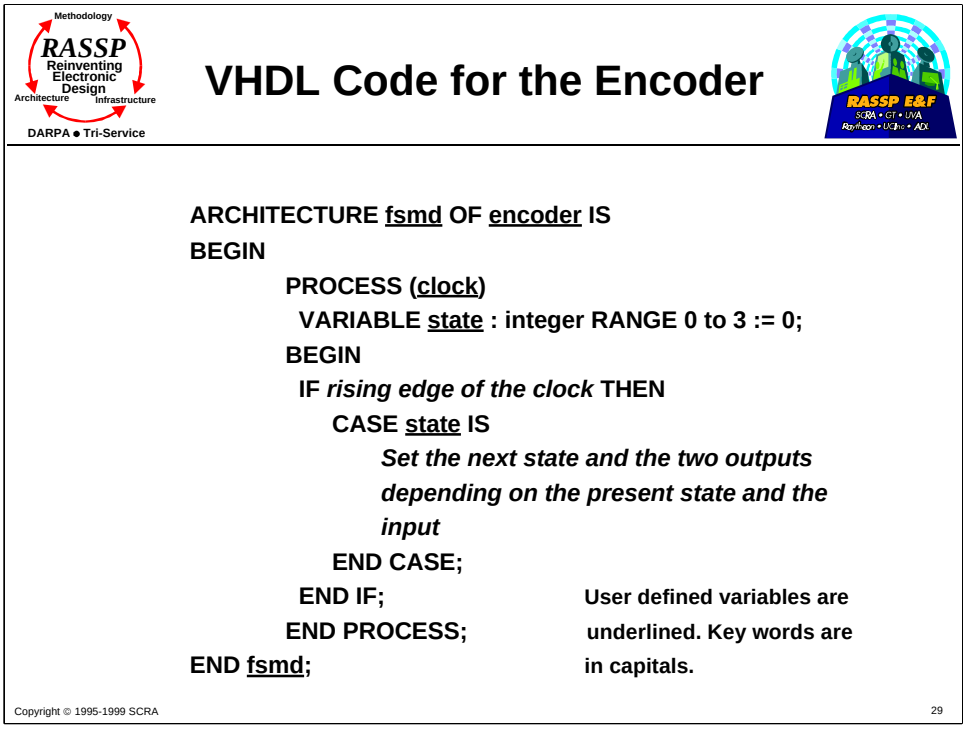


Let us consider a communication system. The input signal x[n] is encoded using a rate 1/2 linear convolutional code. The encoder outputs y0 and y1 are then modulated using binary phase shift keying (BPSK) to produce s[n]. This signal is basically a sequence of -1's and +1's. s[n] is then transmitted over an additive white gaussian noise (AWGN) channel. The output of the channel is demodulated and then quantized using 3 bits. The digitized output of the quantizer is then fed to the Viterbi decoder which generates the final output sequence.

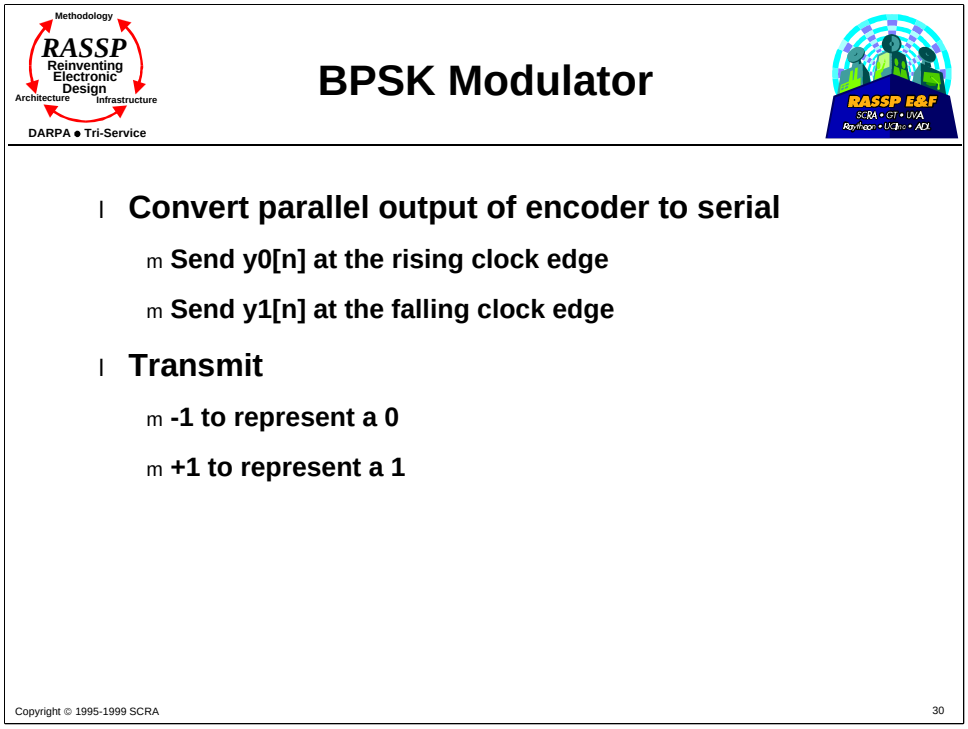
The numbers next to each variable gives the kind of values the variable may take. For e.g., a (0,1) next to x[n] means that x[n] can be a 0 or a 1 and nothing else.



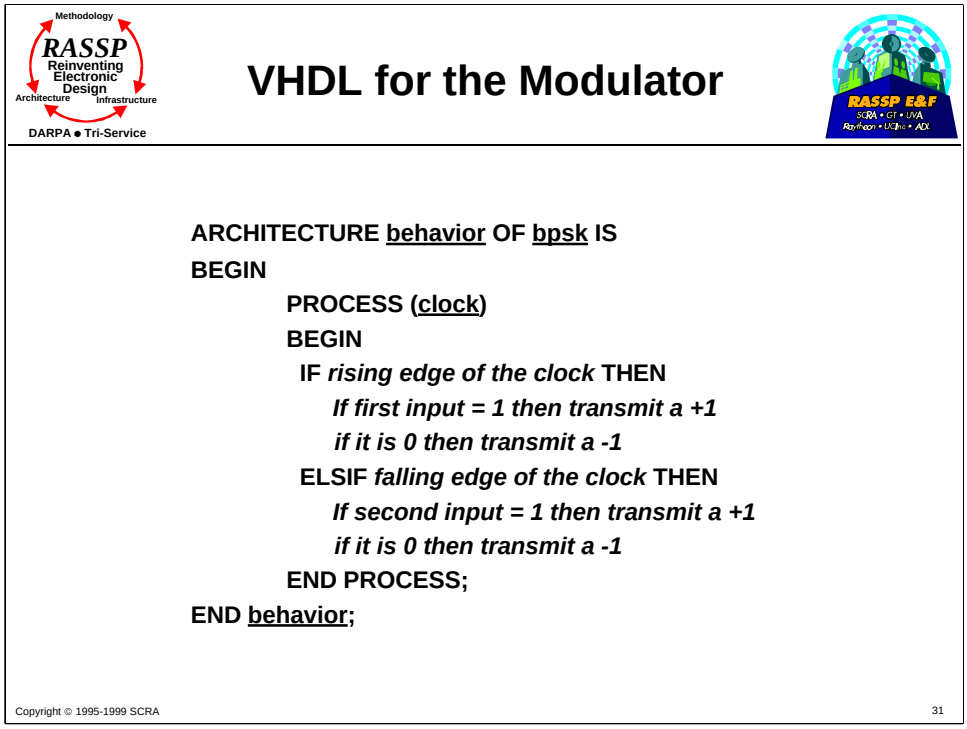
This is a graphical description of the flow graph of a linear convolutional encoder (the input is x[n] and the output is y[n]).



The graphical description is captured in VHDL with clock-level fidelity.



Another example of a commonly used communications subsystem.



VHDL executable requirements of the same BPSK modulator.

AWGN Channel

- **Signal to Noise Ratio (SNR) and Variance**

  $SNR = -10 \log_{10} N_0$

  $Variance = \sigma^2 = N_0$

  $Therefore, \sigma = 10^{-(SNR/20)}$

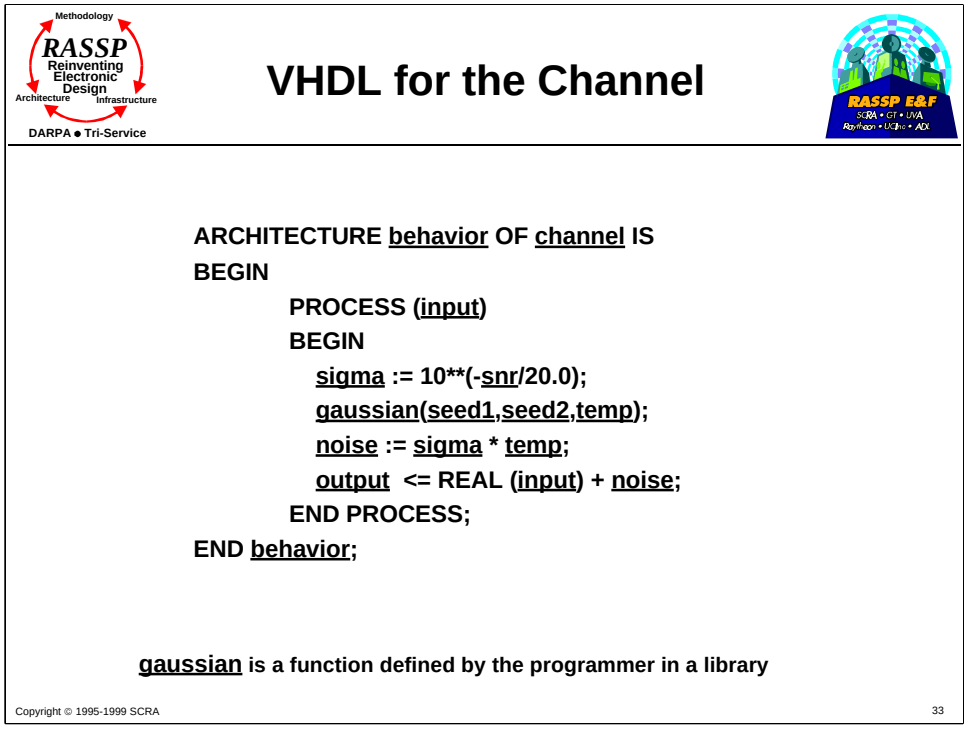
- **Gaussian Noise Generation**

  $x_1 = \exp(-(y1^2+y2^2)/2)$

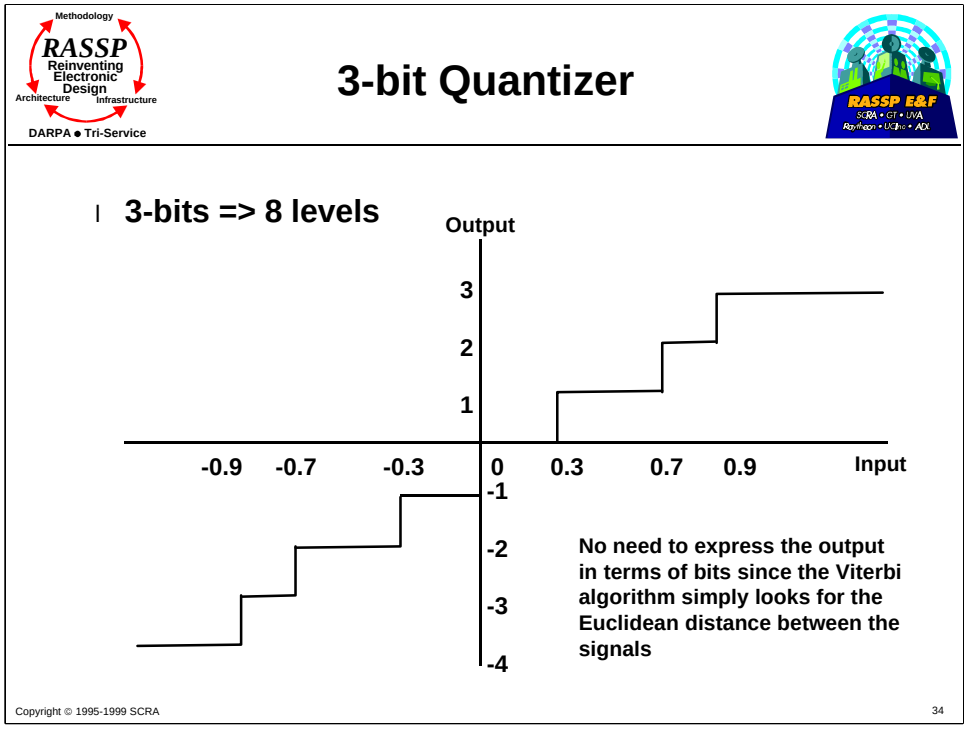
  $x_2 = (\arctan(y2/y1)) / (2\pi)$

  - **Choose x1 and x2 to be Uniform random variables over (-1,1)**
  - **Then y1 and y2 are Gaussian random variables with mean 0 and variance 1**
  - **Scale y1 and y2 to get the right variance**

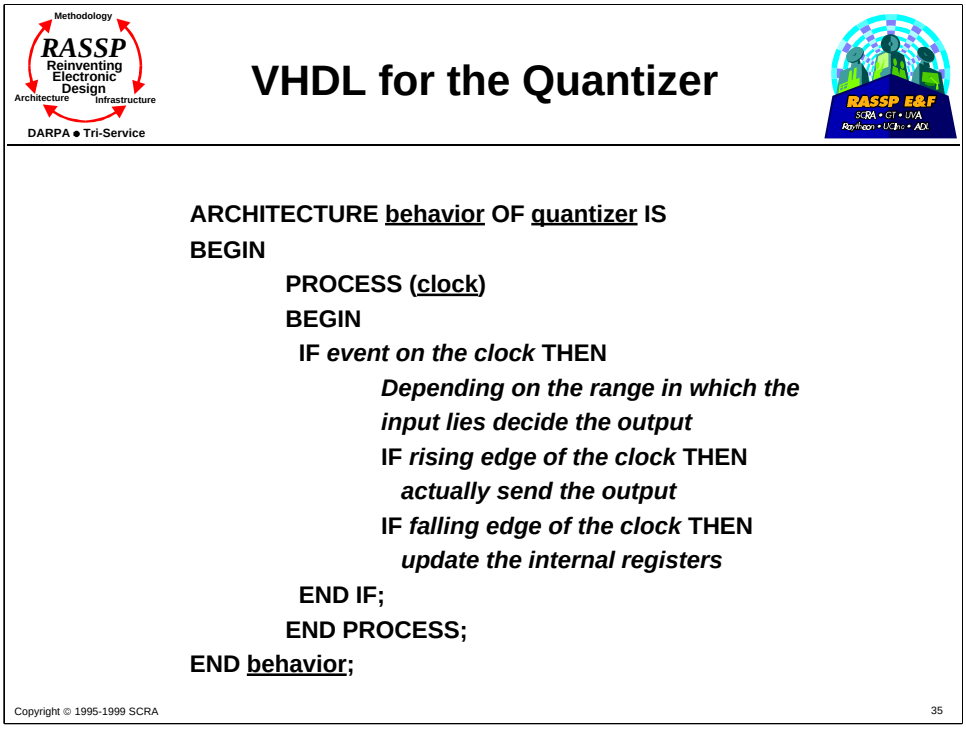
The channel is represented conventionally with specification of the noise and the signal to noise requirements.



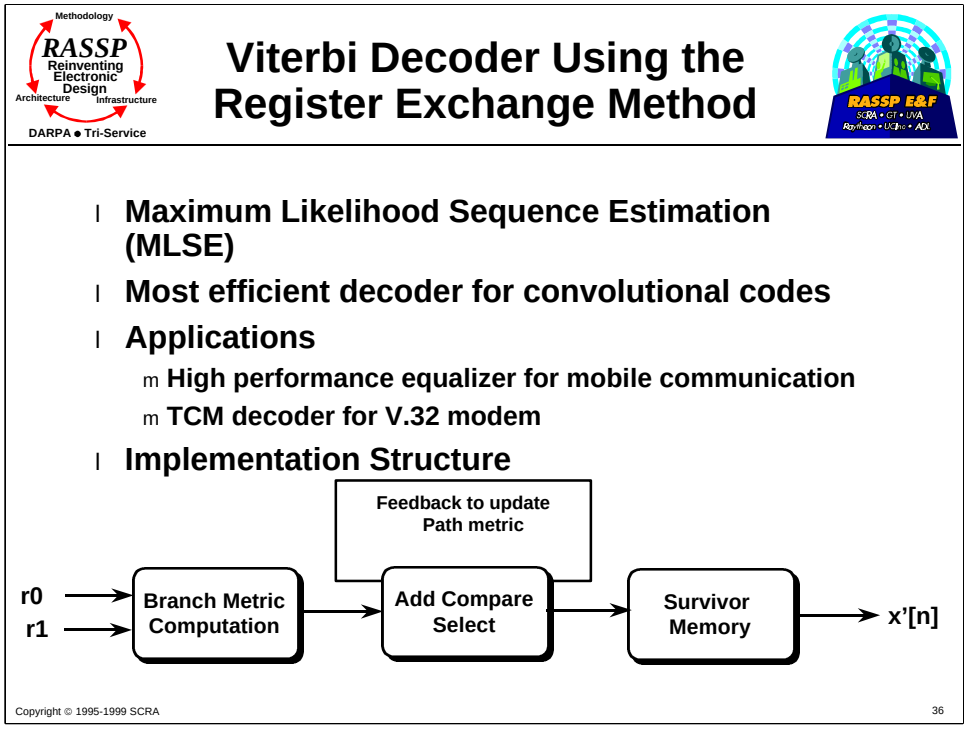
Representation of the channel in VHDL (executable requirements)



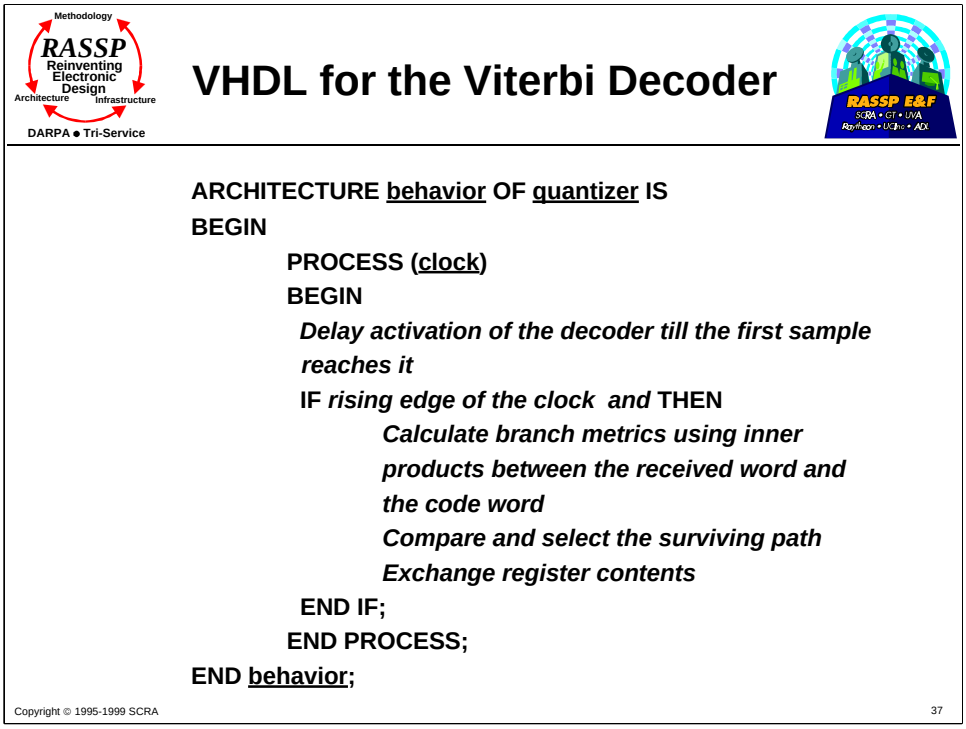
A quantizer is represented in graphical form.



The high level representation (in executable form) of the quantizer.

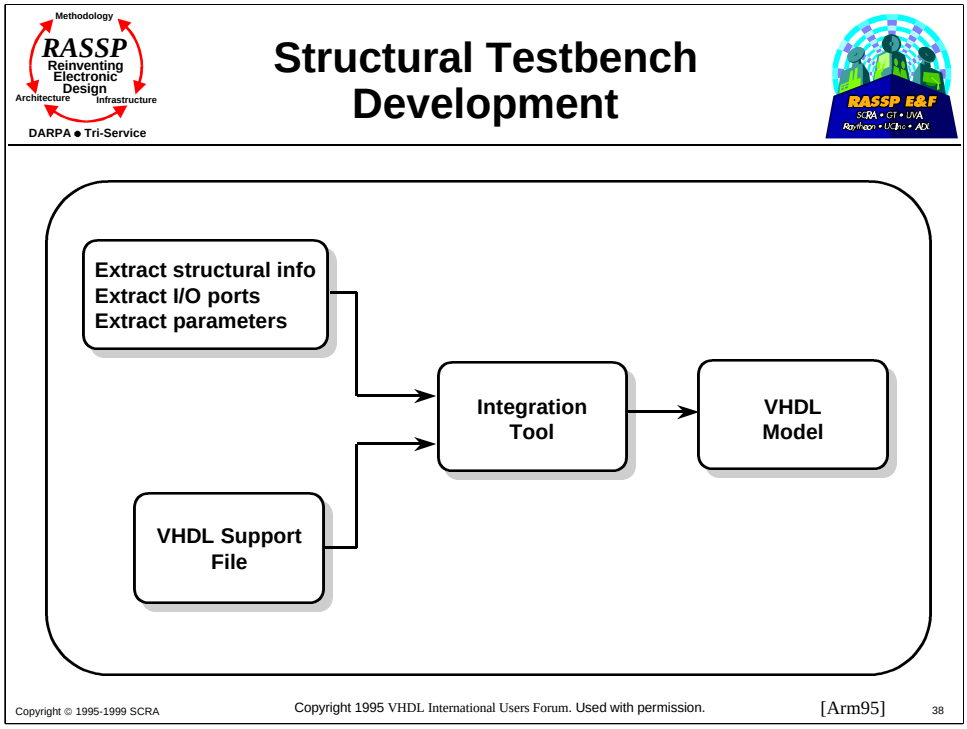


Another commonly used communication subsystem that uses the Viterbi decoder.



Each of these individual blocks are first simulated and tested. Then finally all these entities are connected according to the system block diagram and simulated.

Thus we have been able to test the functionality of an entire communication system using behavioral level specifications. Such simulations help to refine and perfect the algorithm before going to lower levels of design.

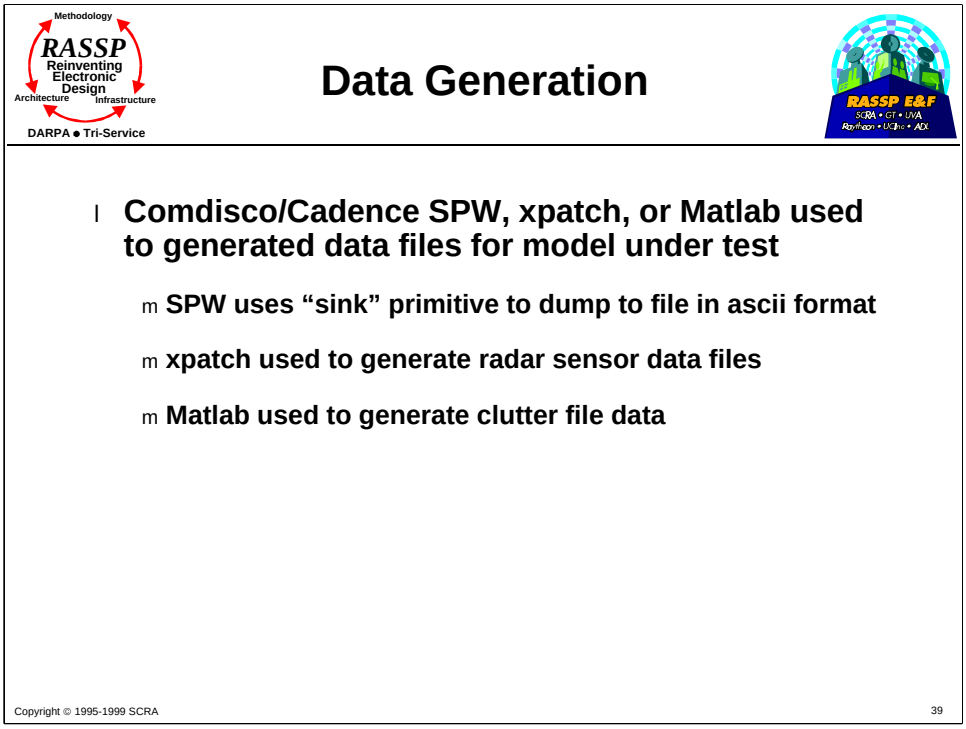


Signal Processing Workstation (SPW) from Comdisco/Cadence was used to create the simulation model of the data flow for the SAR algorithm. SPW can then generate VHDL code from its schematic capture tool. It cannot generate real data types because it only supports fixed-point designs.

A real number model extraction tool that interfaces to SPW had to be developed.

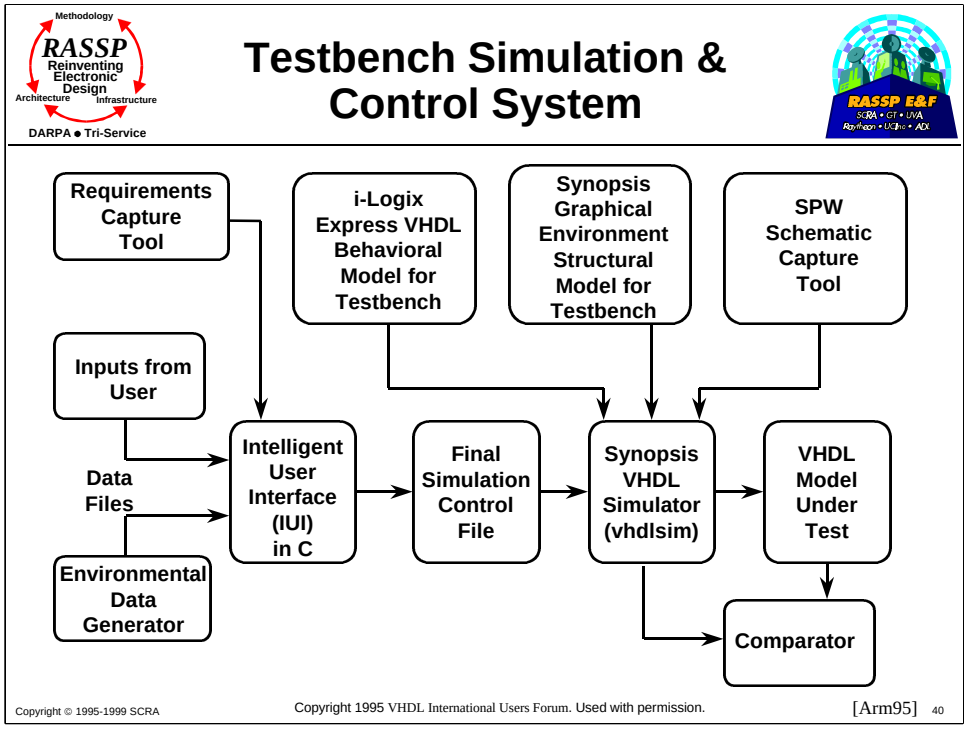
In SPW, every structural model has a schematic description file called “$netlist”. This file contains all the information about the parameters of the blocks and connection information. The above figure shows the methodology of the SPW tool to do this.

The Integration tool is used to integrate all the structural information, I/O information, and parameters captured by shell scripts. It also uses a VHDL support file which is a package of library VHDL procedures corresponding to SPW primitives.



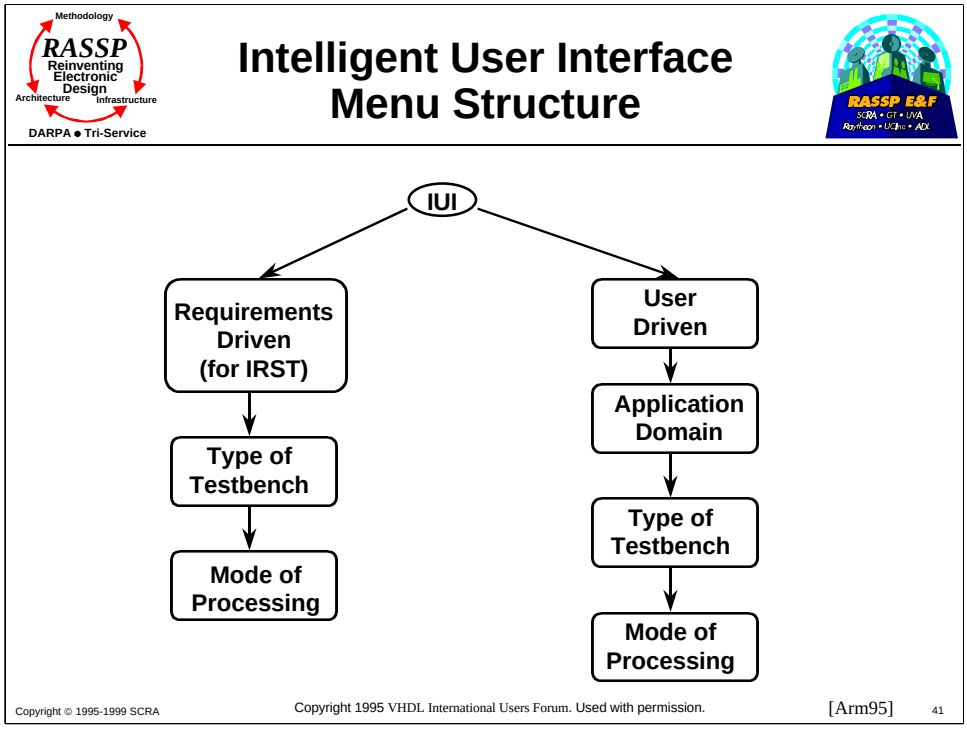
[Arm95]

Algorithm design tools (Matlab, etc.) can be used to generate numerical testbenches for use in system design environments.



This shows a combination of all the previous techniques mentioned as part of a complete environment for testbench generation and model simulation.

The next slide describes the IUI in more detail. Note that recent languages such as The Unified Modeling Language (UML) can also be used with advantage (just like VHDL has been used in this module).



The main purpose of the IUI is to configure a testbench that is appropriate to a test or a set of tests, given that all the VHDL components needed for the testbench are available.

The system requirements can be applied to the testbench model through the IUI. It can also generate the necessary input data values (target initial position, etc.) for the testbenches by either user interaction or through file I/O.

The IUI forms the simulation control file for the simulator. This file is used to create displays during or after simulation of the testbench and controls the overall operation of the simulator.

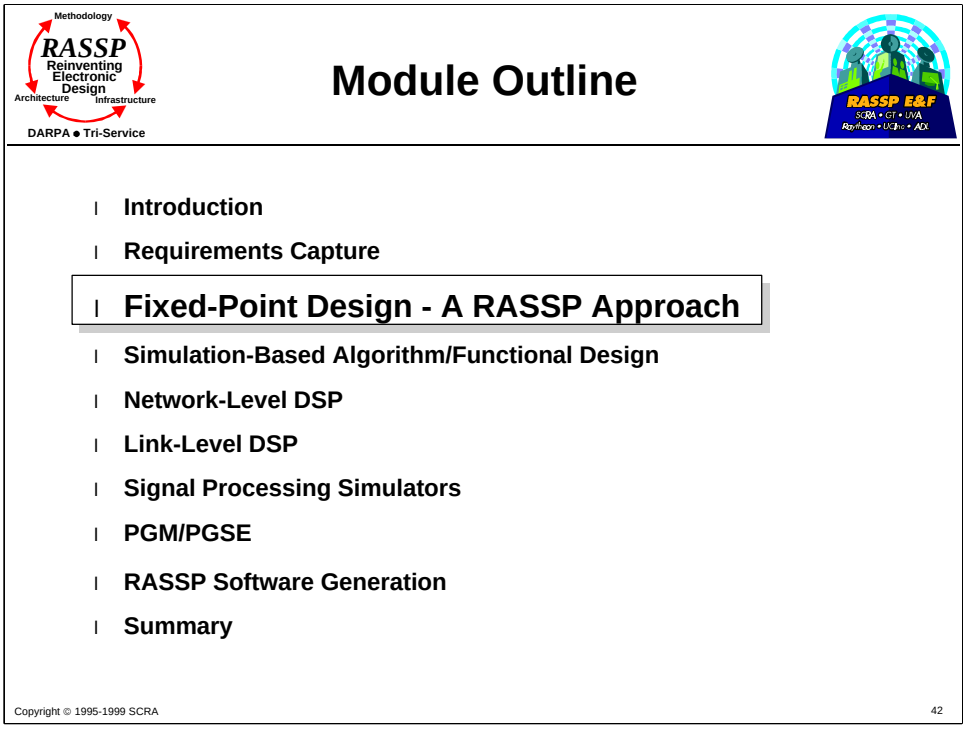
The IUI forwards the testbench outputs to the model under test for testing that model.

The IUI was developed in C and runs on a Sun-OS platform.

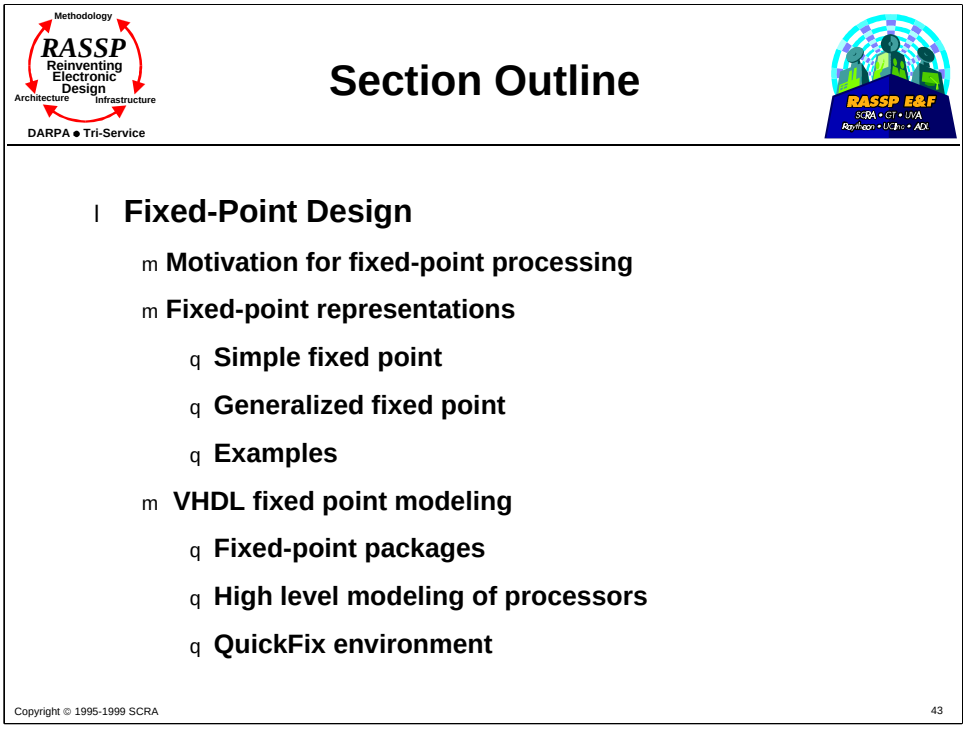
The menu structure is shown in the above diagram.

It provides the user with a choice of categories such as application domain, type of testbench, etc.

Based on all input values gathered by the IUI, a simulation control file is formed to control the simulation of the the testbench model.

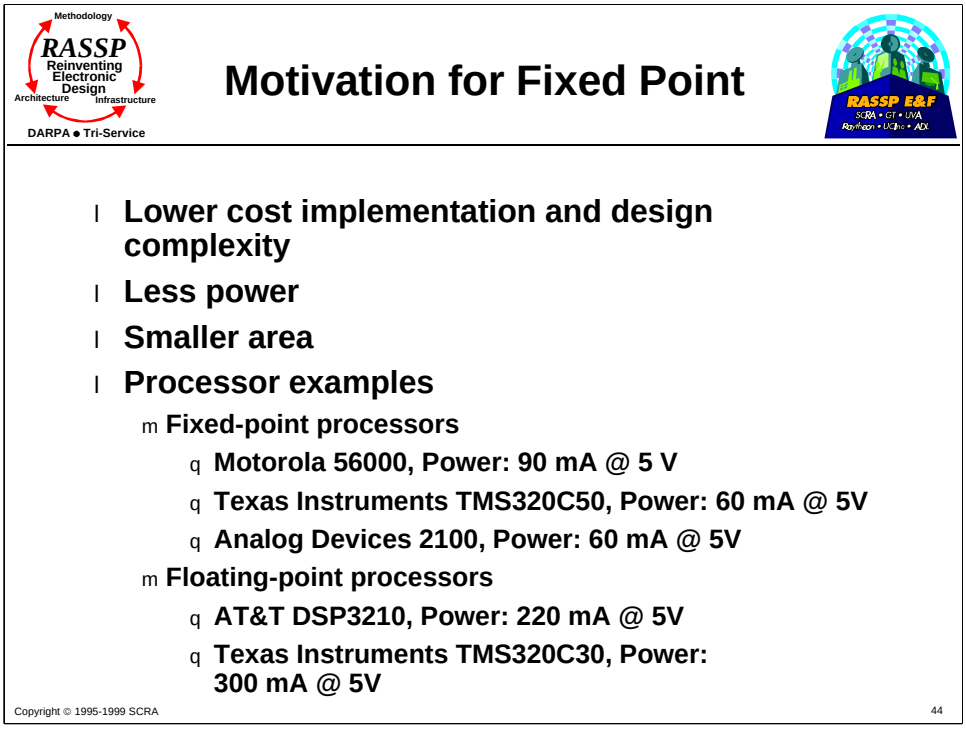


After the high level simulation of the requirements, it is often necessary to refine the requirements one step further within a fixed point simulation environment.



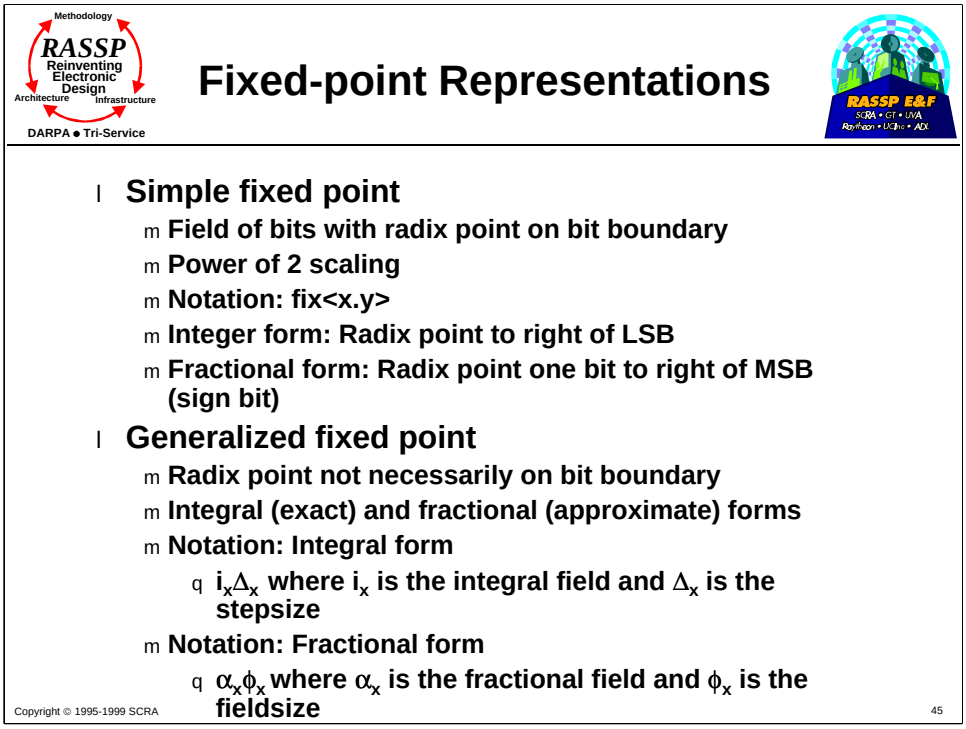
This section will present an overview of modeling and simulating fixed-point representations of algorithms.

Typically the algorithm is simulated in double-precision floating point in an environment such as Matlab. After the performance is studied at this level, fixed-point implementation can be explored to save on hardware cost and power consumption.



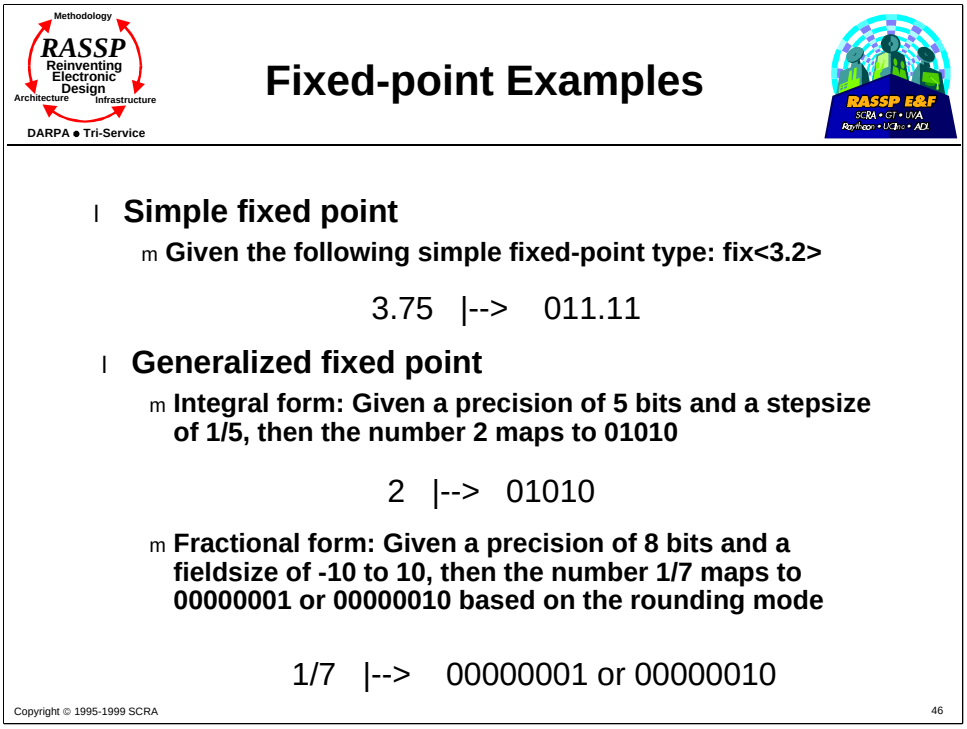
The above slide lists some reasons for using fixed point processing when the requirements for the algorithm can be met with this accuracy. The power and size are less when compared to floating point, and the complexity tends to be less.

What is needed is a good front-end tool to help make rapid tradeoffs between specific floating-point and fixed-point implementations because most simulations are done in floating point. It takes more time and effort to simulate in fixed point.



[Baudendistel93]

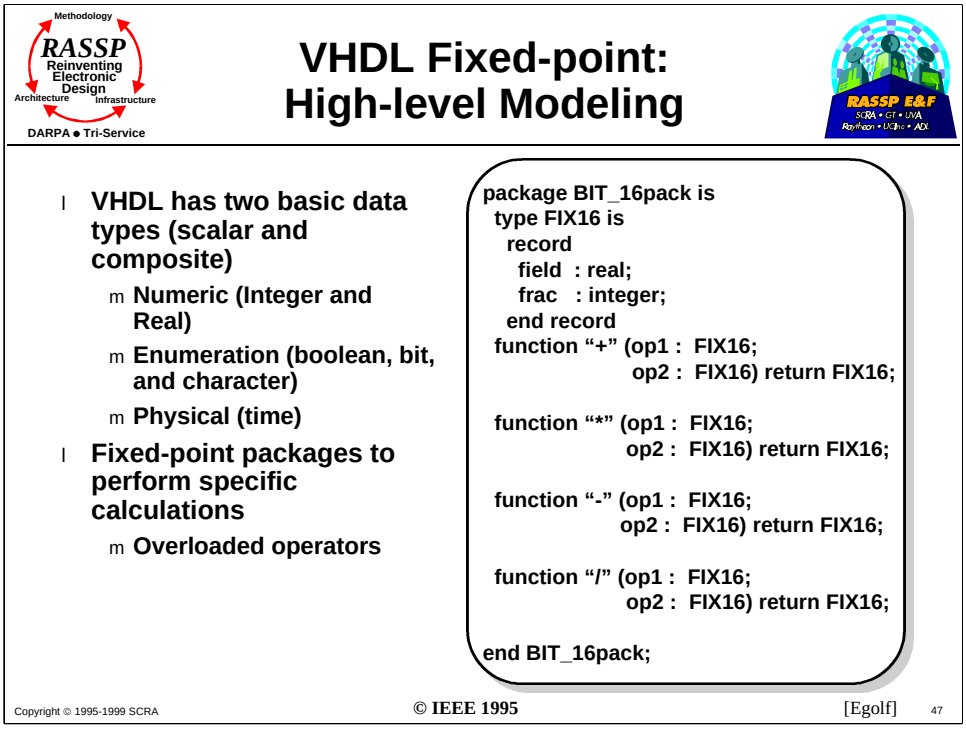
Fixed point can be represented by either the simple form or a more generalized form as shown above. Most simulators of fixed point use the simple form (Ptolemy, Mentor Graphics, Virtuoso). The generalized form is more difficult to implement but provides benefits when it is preferable that the radix point not lie on a bit boundary.



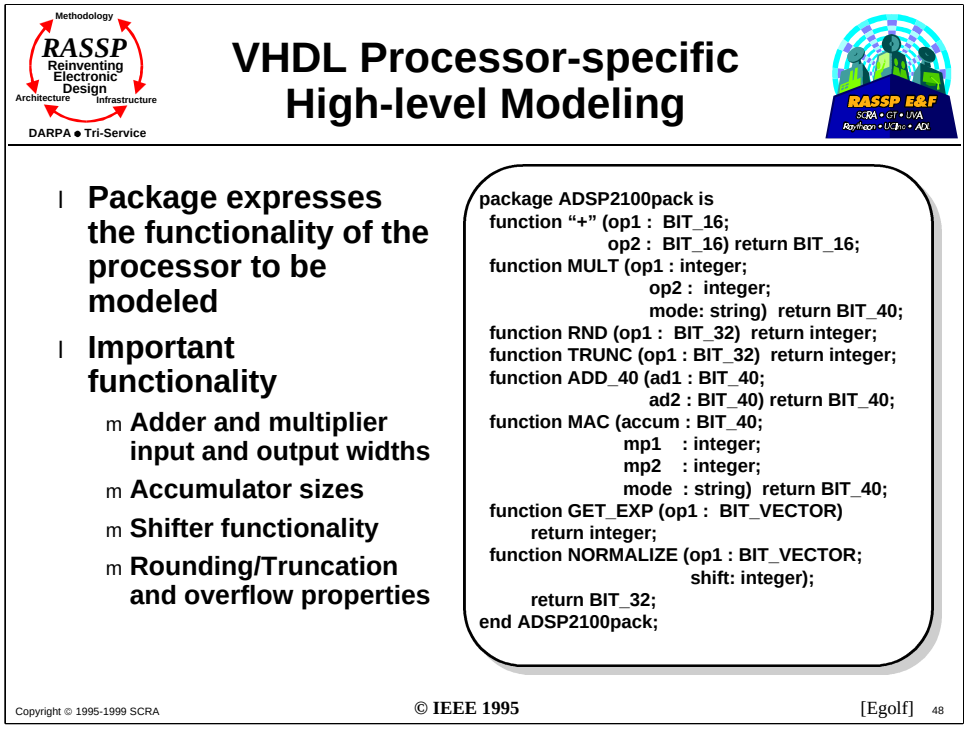
This slide shows an example of both the simple and generalized fixed-point representations.

The simple form represents the radix point on bit boundaries and, as can be seen, 3.75 maps exactly to the value shown using the fix<3.2> format. This is the maximum value represented by this format because the first bit is the sign bit. The resolution is .25 and numbers such as 3.2 cannot be represented exactly.

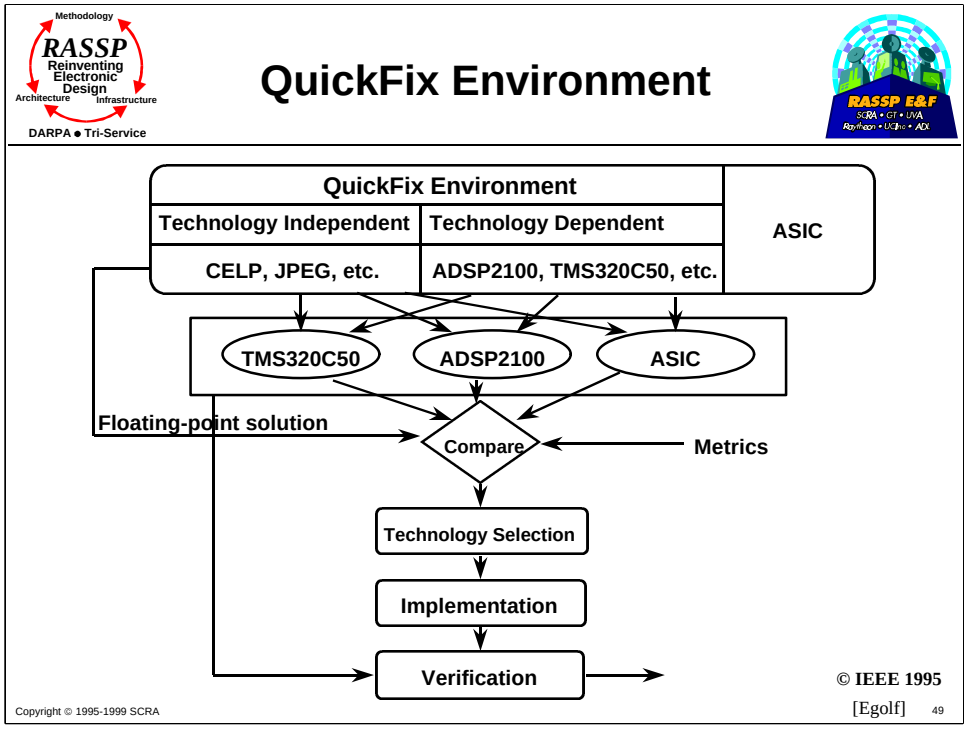
The generalized form specifies numbers with a slight difference. The precision and stepsize are used for the integral form, and the precision and fieldsize are used for the fractional form. Although the same number of numbers can be represented in both when the precision is the same, there is a slight difference in interpretation. In the integral form all numbers are represented exactly, while in fractional form approximations are used to represent specific numbers, as can be seen from the examples. The fieldsize represents a range of possible real numbers, while the integral form represents a range of integers.



This is how the fixed-point representation can be done in VHDL. The use of operator overloading can help the developer define the operations on this scaled-fractional data type. Additional operations besides the ones listed above must also be included, such as greater than, less that, negation, and not equal to.



Using the same concept as in the last slide, we can also use VHDL to make processor-specific packages. These mimic the behavior of a specific fixed-point processor by representing its internal functionality in a package with procedures and overloaded operators that behave exactly like the processor. As we can see from above for the Analog Devices 2100 fixed-point package, there are functions to model its accumulator, adder, multiplier, shifter, and its rounding or truncation behavior.

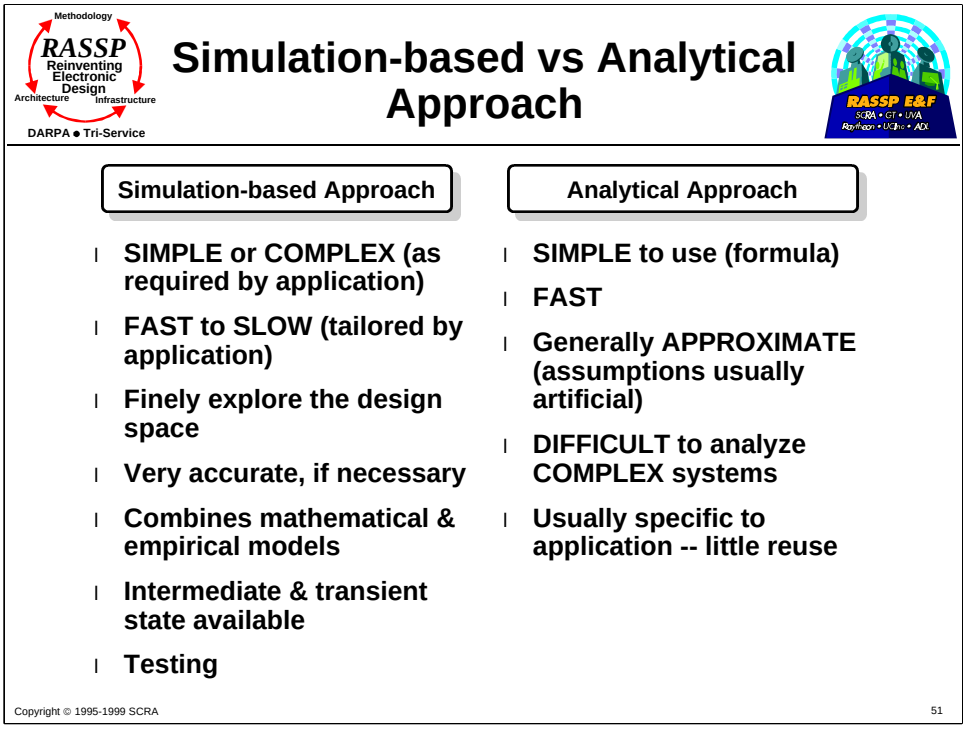


When all the previous methods are combined, we arrive at an environment in VHDL that allows a algorithm designer to quickly examine the behavior of the algorithm using various bit widths to examine its performance requirements.

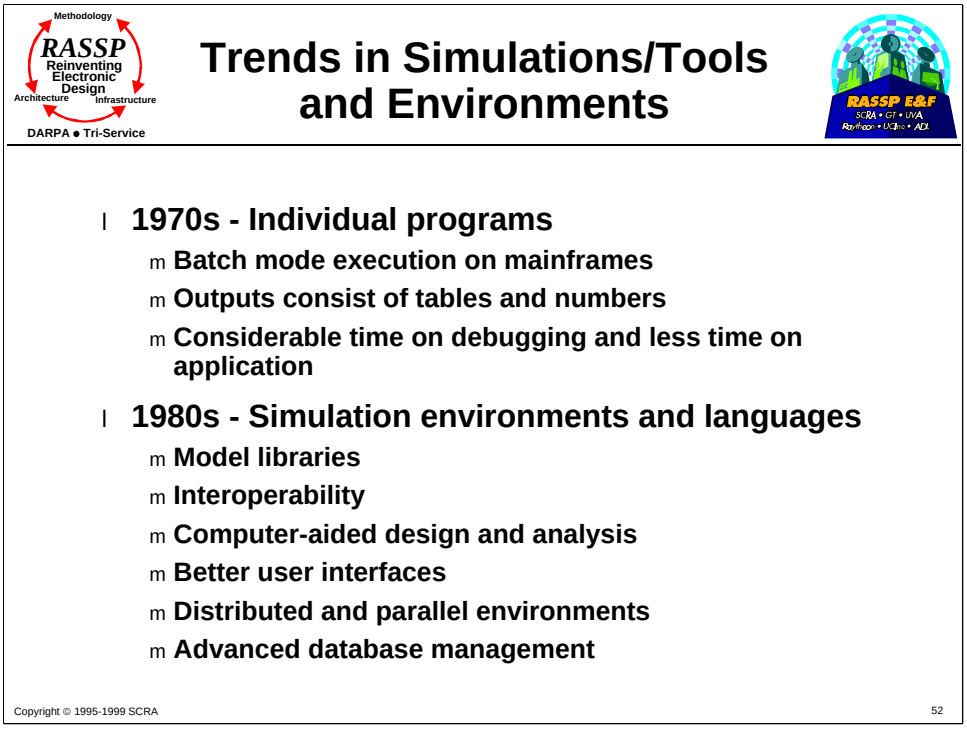
This can then point the designer to the correct architecture to use to implement the algorithm's behavior.

Module Outline

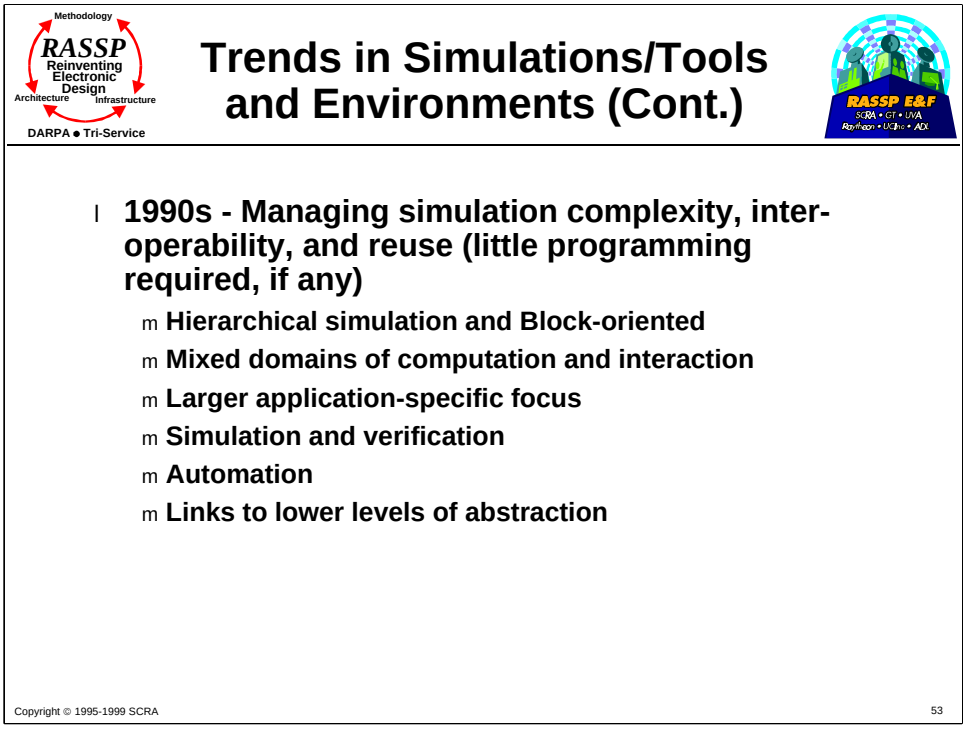
- Introduction
- Requirements Capture
- Fixed-Point Design - A RASSP Approach
- **Simulation-Based Algorithm/Functional Design**
- Network-Level DSP
- Link-Level DSP
- Signal Processing Simulators
- PGM/PGSE
- RASSP Software Generation
- Summary



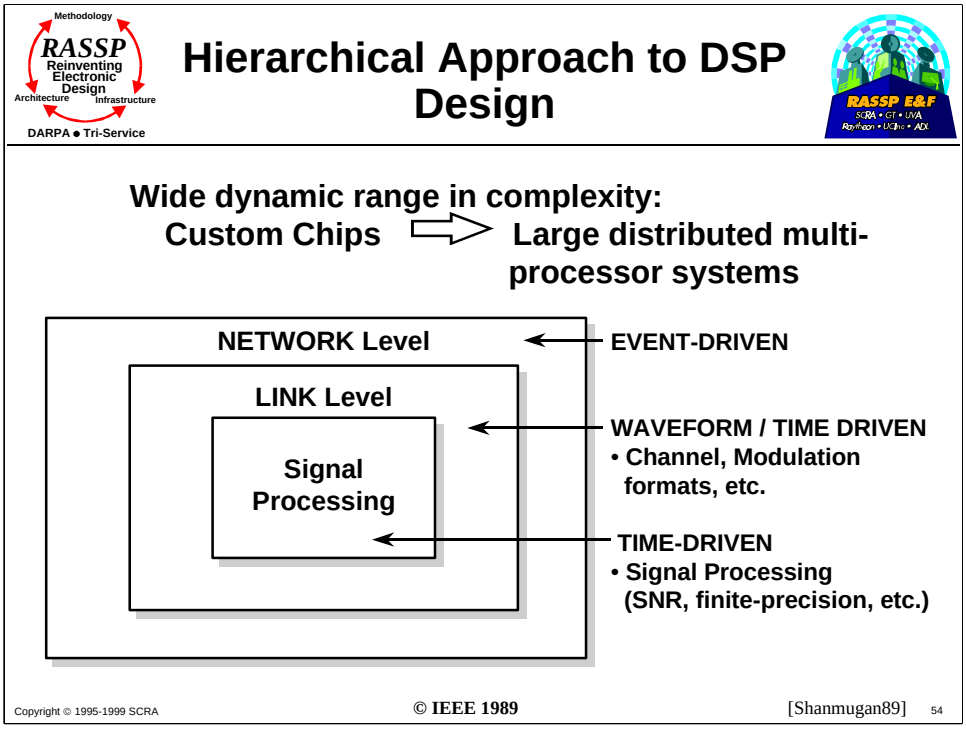
Here we compare the advantages of a simulation-based design methodology with respect to an analytical approach.



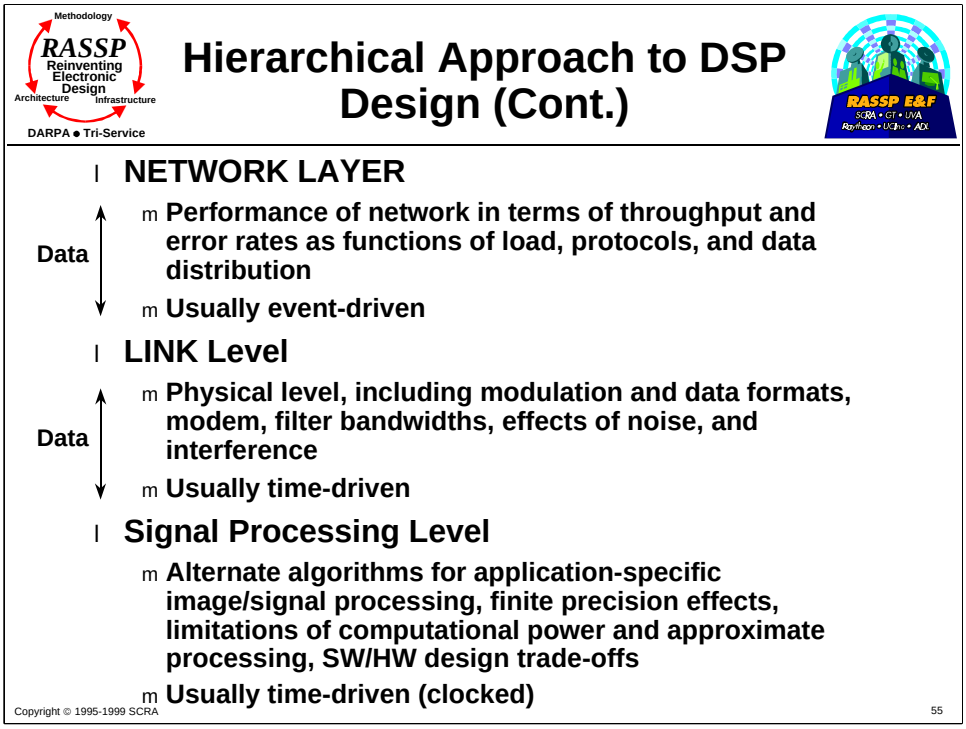
This slide describes the progression in simulation-based design environments.



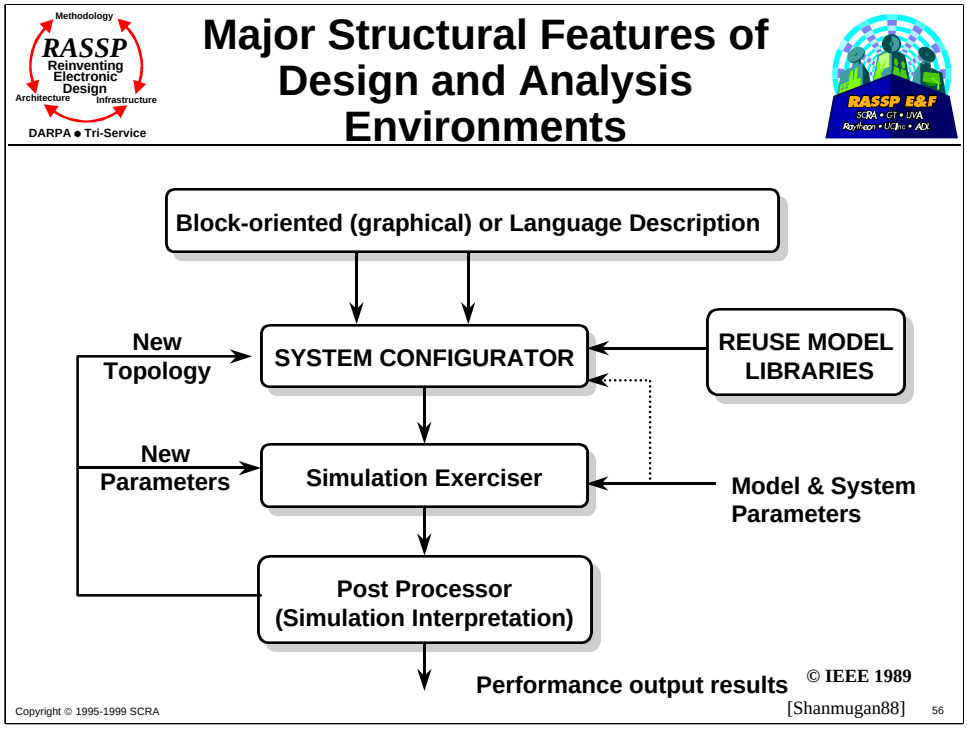
The trends for the 90's are listed above. The Unified Modeling Language (UML) is a late 90s evolution that promises to unify the advantages of various modeling paradigms.



We find this convenient as a unified picture of the “layered” DSP algorithm design environment. We will discuss the different environments in the context of this slide.



This slide describes the “heterogeneous” or “mixed domains of computation” found in modern environments such as Comdisco, COSSAP, and Ptolemy.

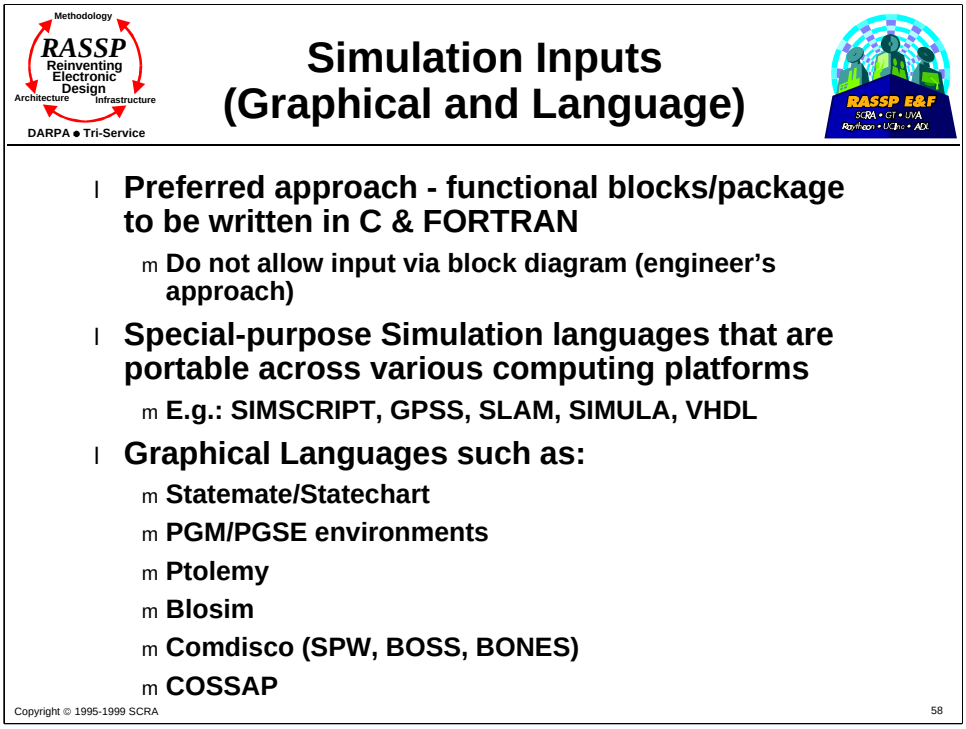


This slide breaks down the design environment into its functional components.

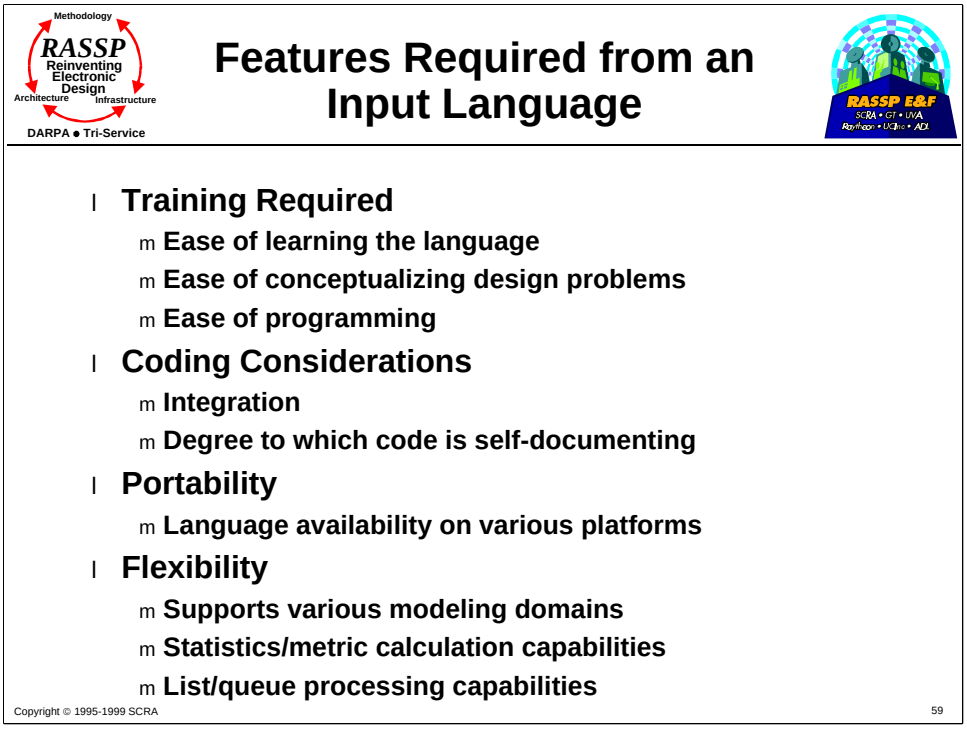
Functional Tasks

- **System Configurator**
  - **Selects functional blocks from model library**
  - **Connects them in the desired topology**
  - **Can set parameters**
- **Simulation Exerciser**
  - **Sets up the execution of the simulation**
  - **Schedules, assigns, and allocates resources**
  - **Generates and uses testbenches**
  - **Performs checkpointing and storage of simulation state**
  - **Performs garbage collection and recovery**
- **Post Processor**
  - **Examines time & event histories**
  - **Computes performance metrics**
  - **Formats and displays interpreted results**
  - **Decides on the termination conditions (iterations)**

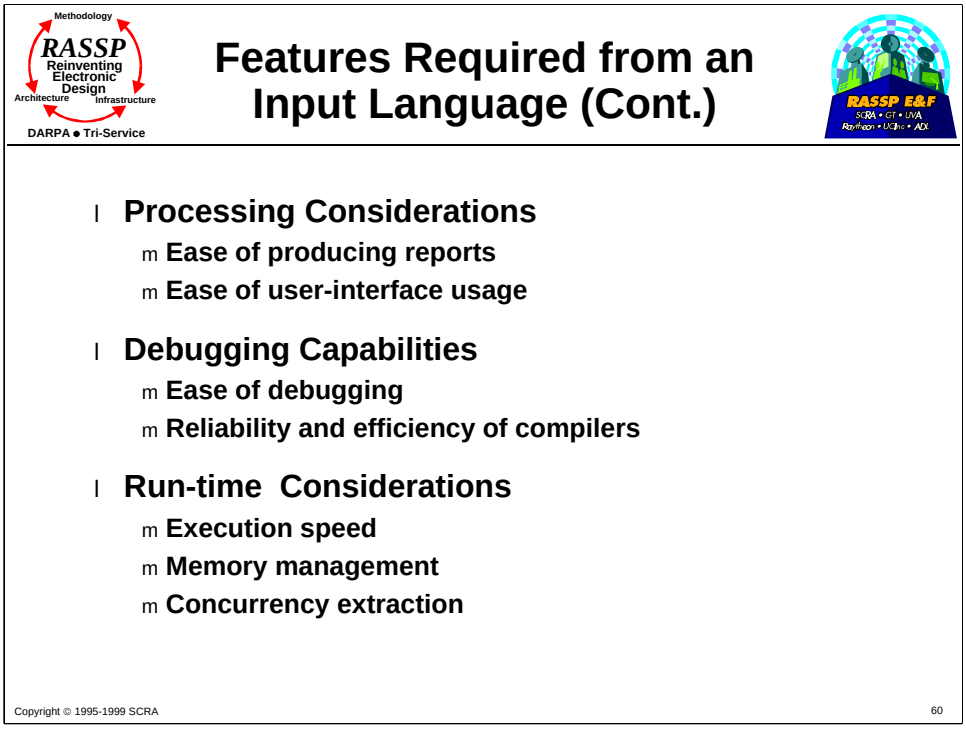
The functional tasks of a simulation-based environment are described.



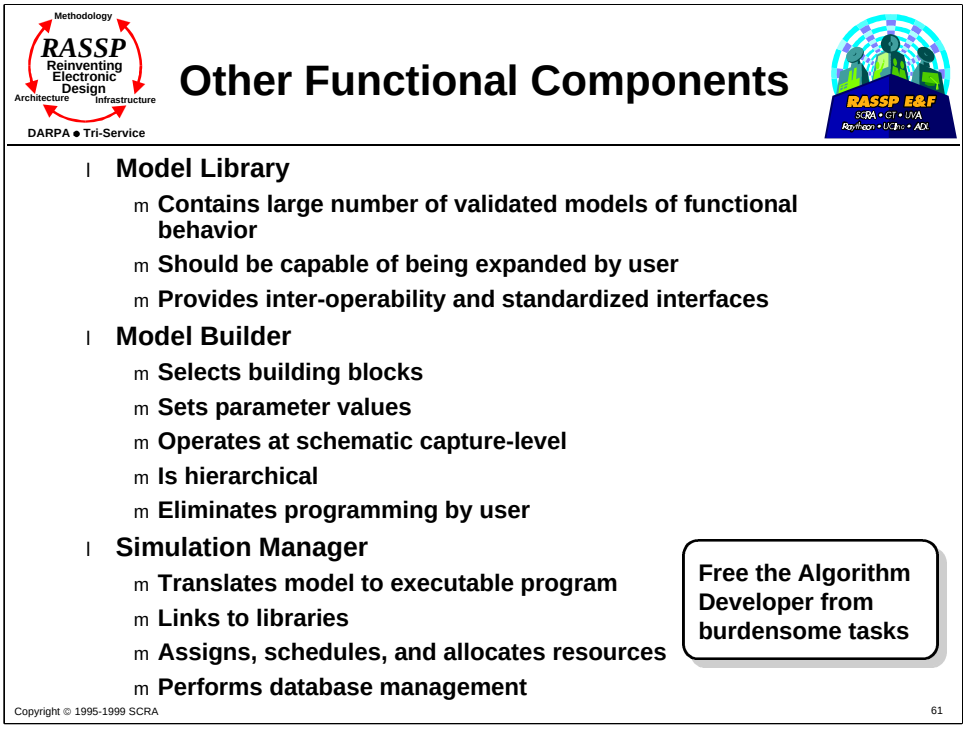
The typical tradeoffs TO BE MADE between graphical AND language-driven approaches.



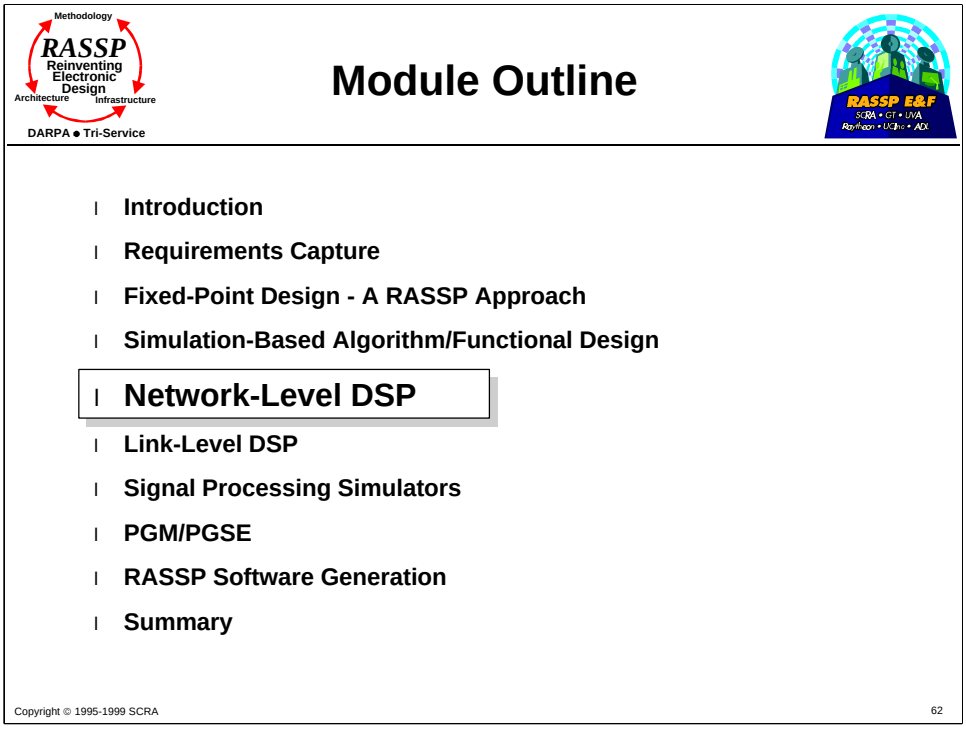
Please refer to The Unified Modeling Language (UML) site at www.rational.com/uml/index.jtmpl



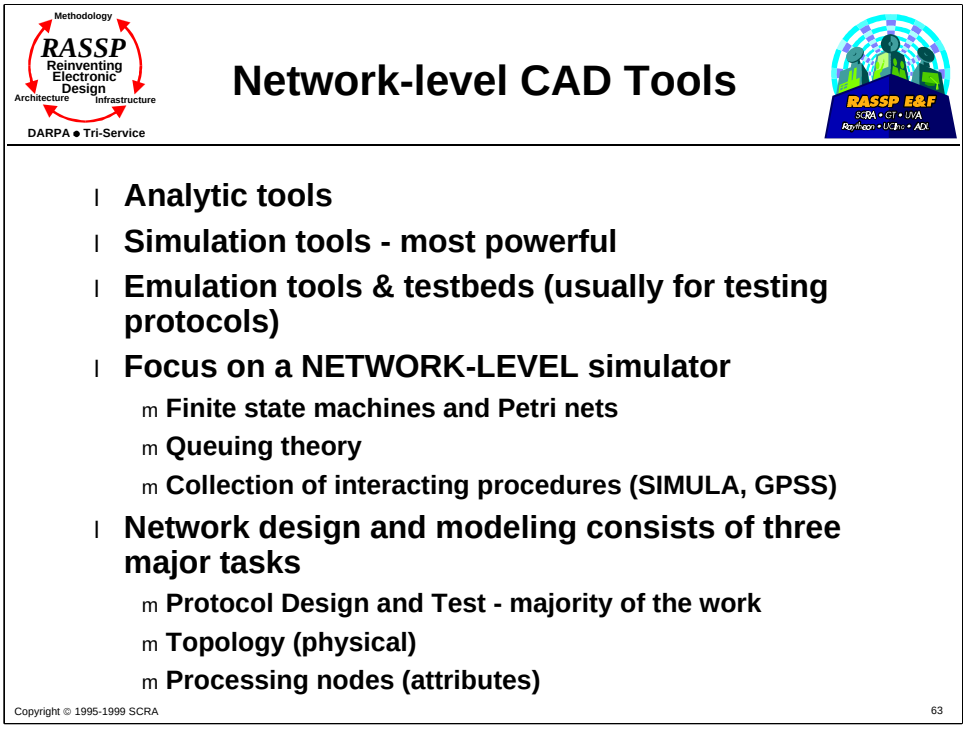
Features of languages (Contd.)



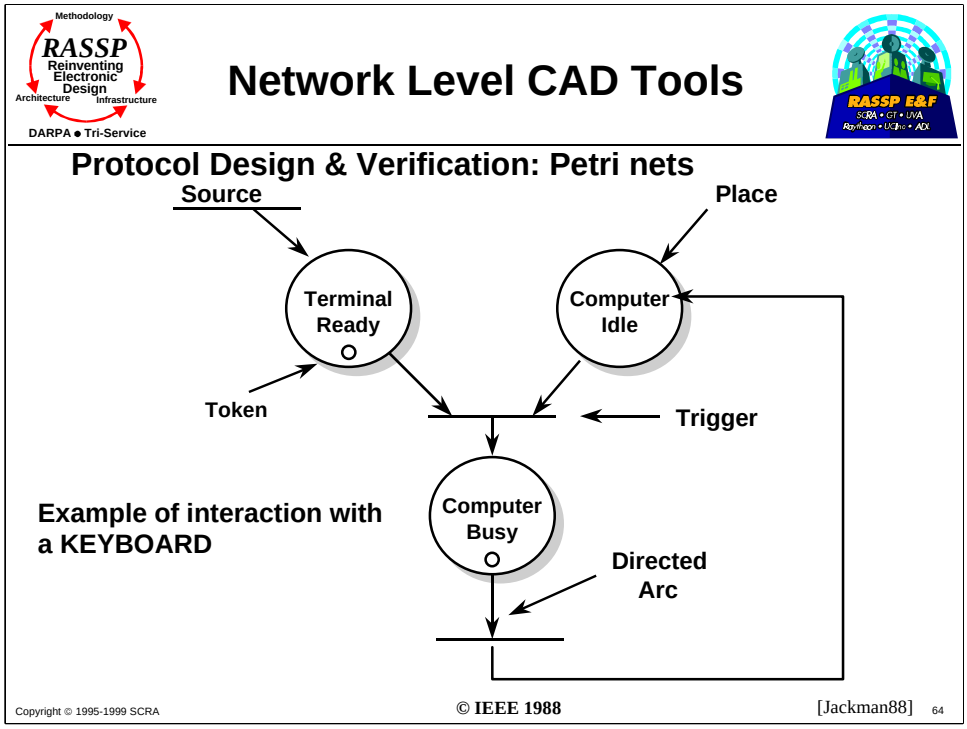
RASSP emphasizes reuse and library generation, and management is a crucial component of the design environment.



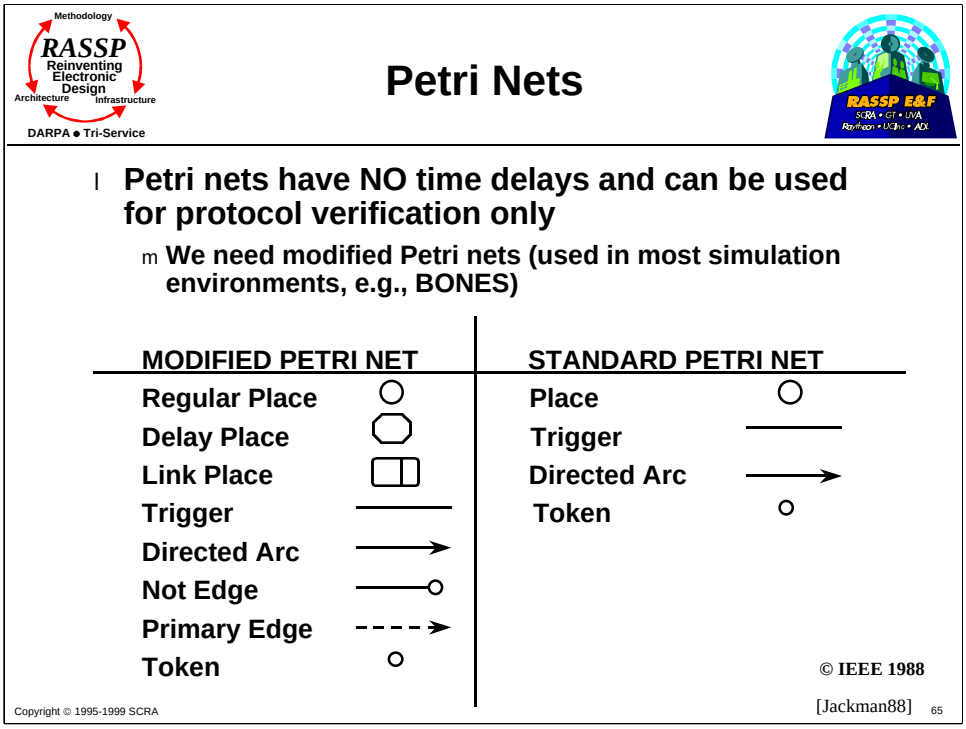
We now progress to modeling a DSP algorithm at a higher level of abstraction - the network level.



We now describe the highest layer of abstraction - network level.

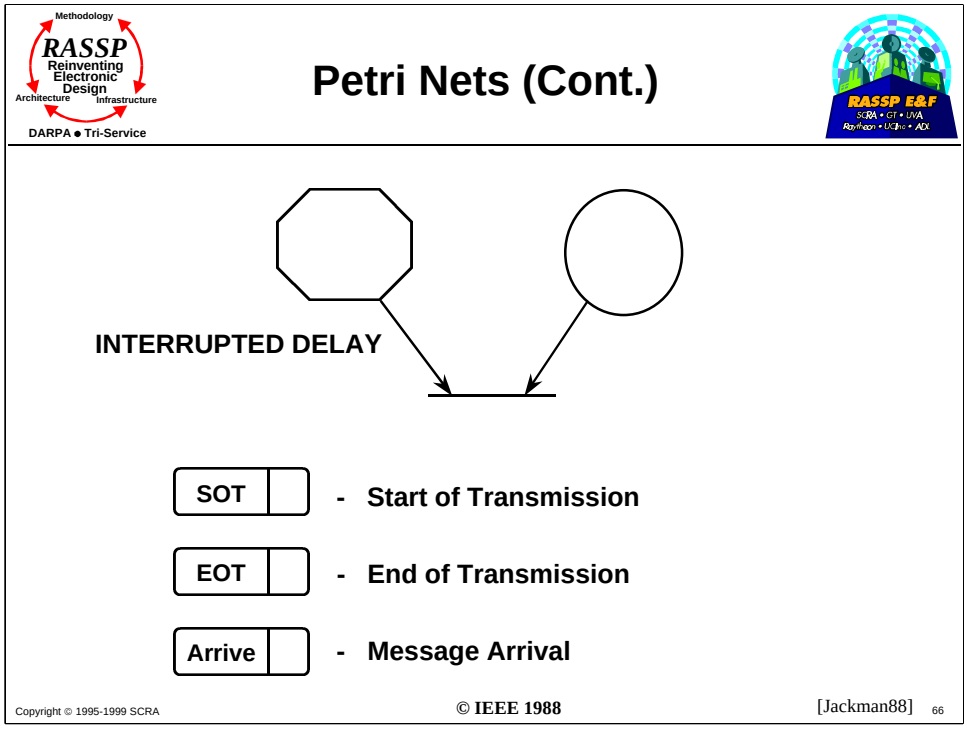


This example shows that one can't use the CPU from the terminal unless the token enters the "idle" state.

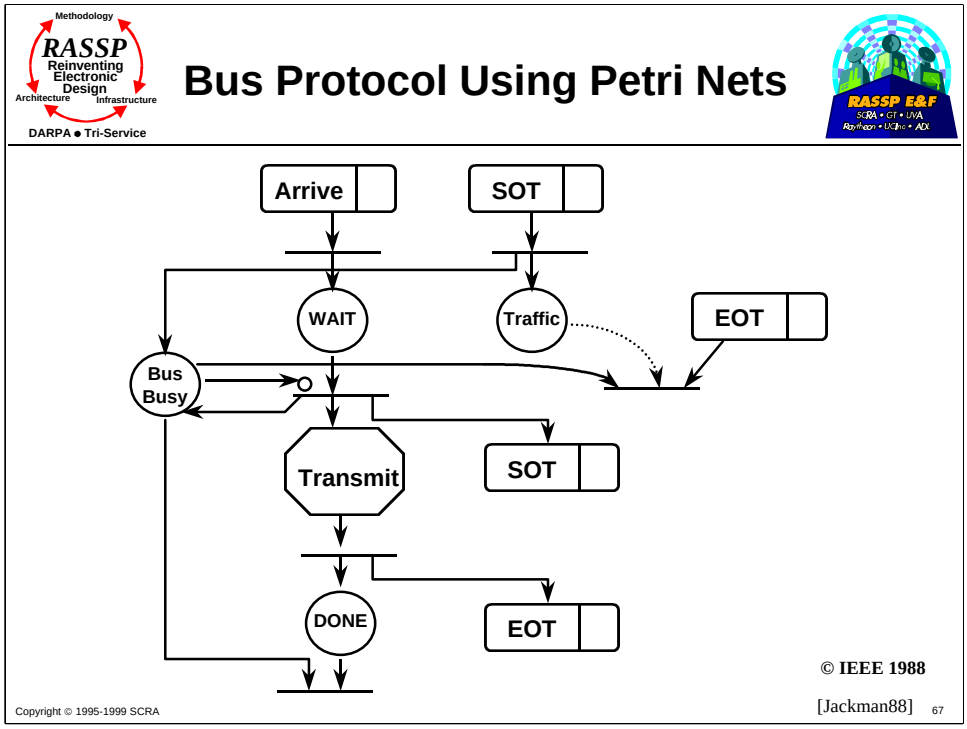


Conventional Petri nets have limitations.

These are improved by the so called “modified Petri net”

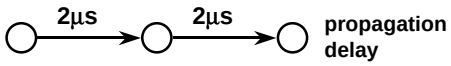


Explanation of some of the features of modified Petri nets.



Example of the use of modified Petri nets in timing simulation in addition to protocol verification.

Detailed Example



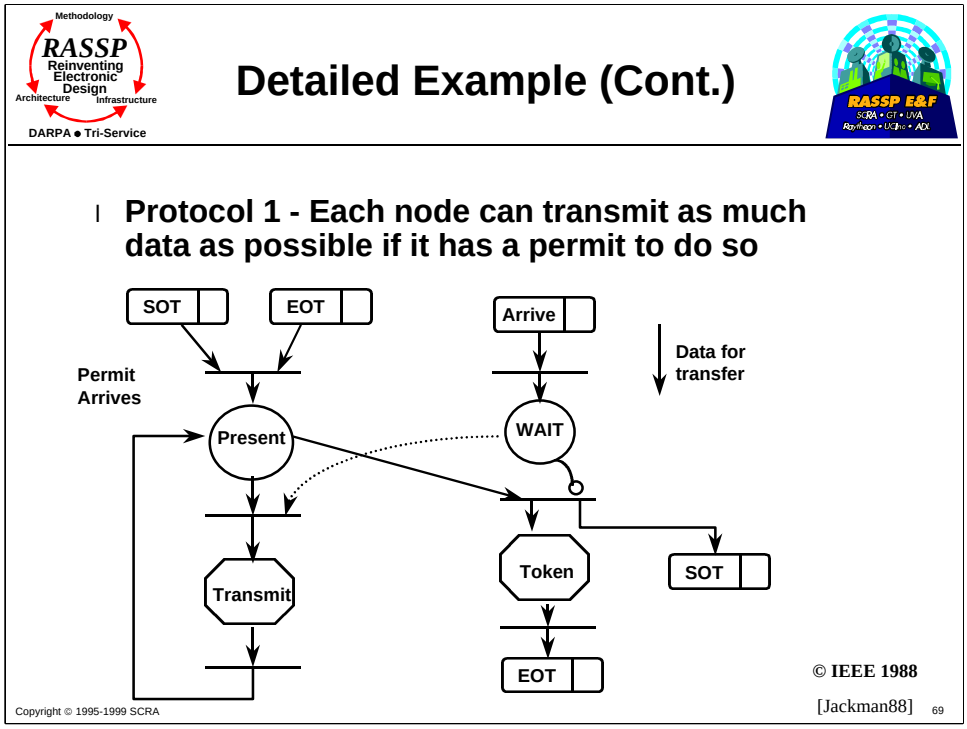
- **Topology**
  - **Physical connections**
  - **Propagation velocity of the medium**
  - **Transmission rate of the medium**
  - **Transmission type**
- **Node**
  - **Arrival distribution of packets**
  - **Packet-length distribution**
  - **Location of node on the network**
  - **Traffic pattern**
- **Protocol**
  - **Routing table for source/sink configuration**

delay in µs

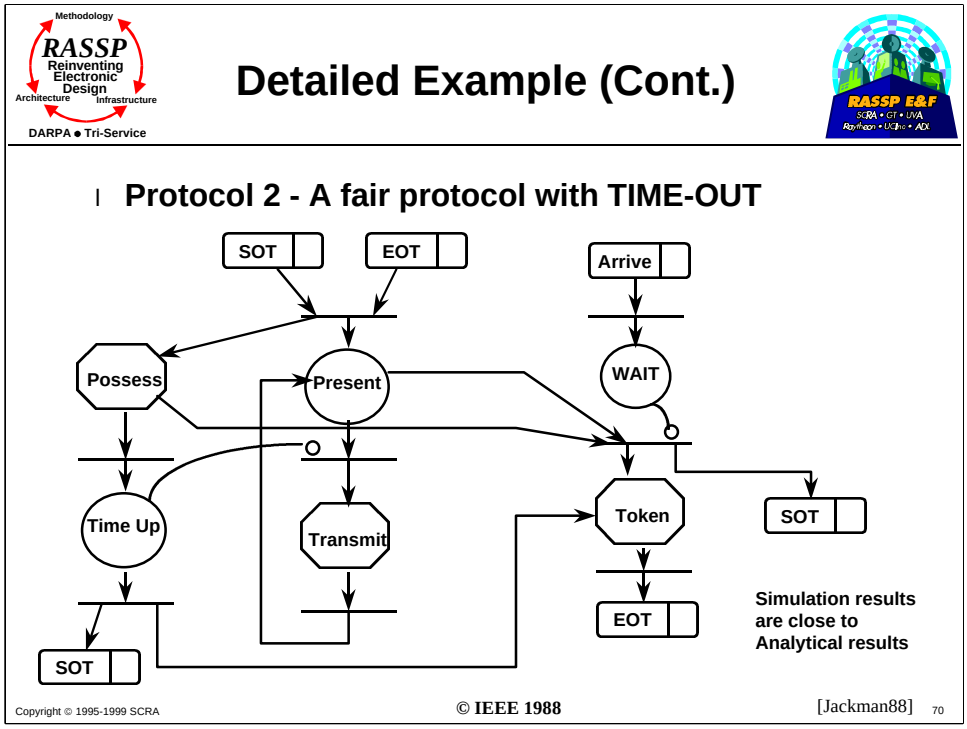
From/To	1	2	3
1		4	
2			4
3	6		

© IEEE 1988
[Jackman88]

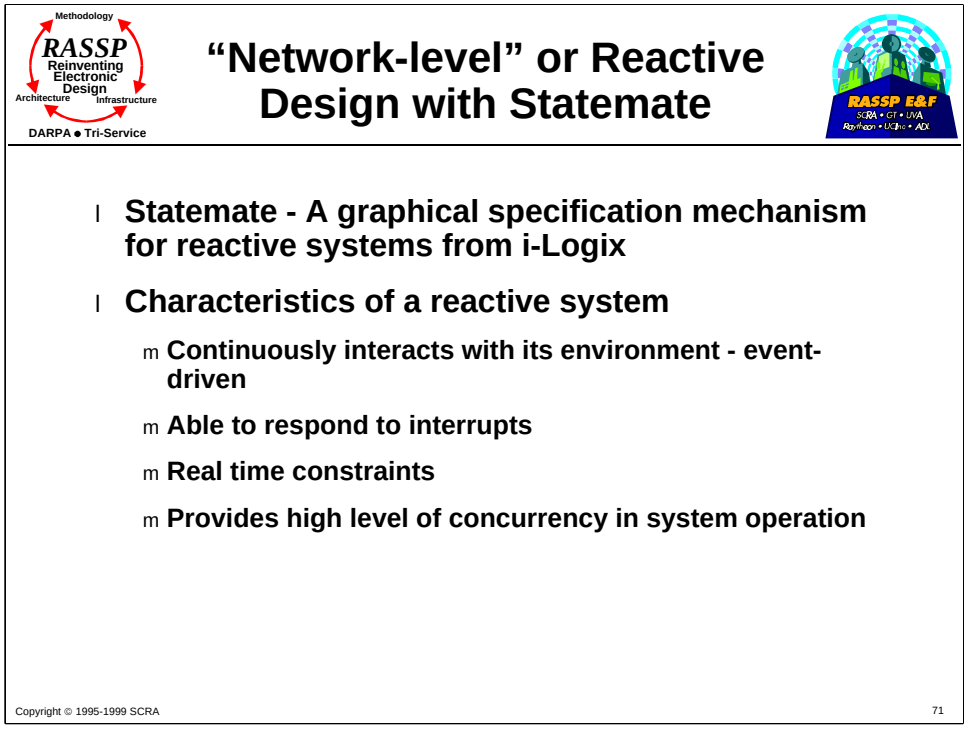
We will now discuss the design of a complex protocol of a multiprocessor network design.



The first example shows the limitations of a simplistic protocol - possibly unfair.



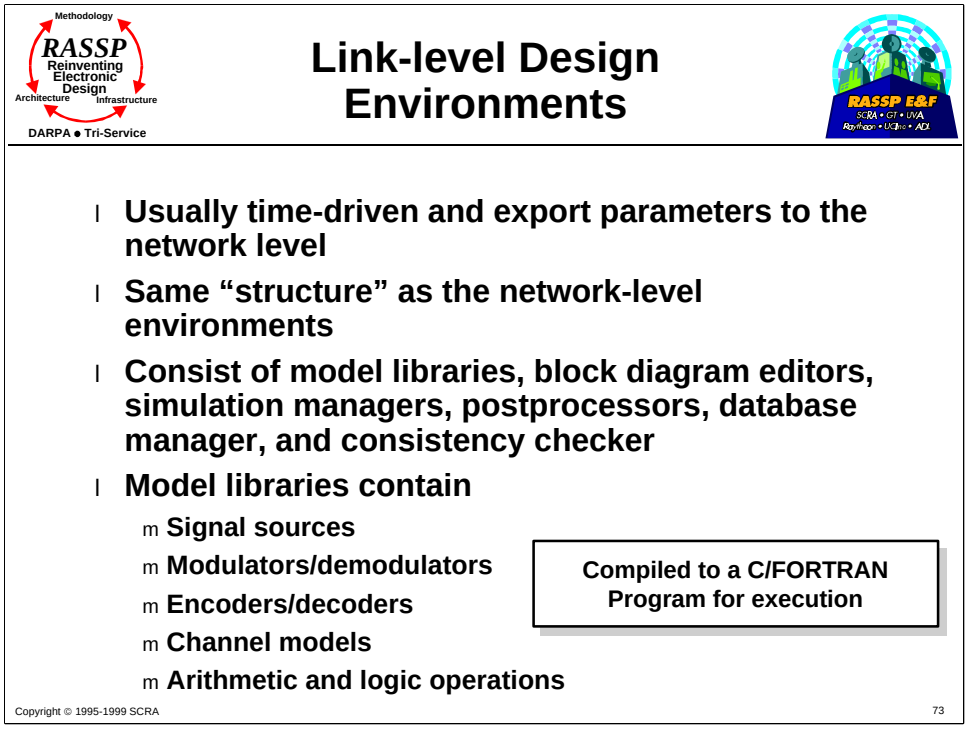
We include the provision of a “time-up” that puts an upper bound on the “hold” time of each node. Actual input load characteristics can then be simulated.



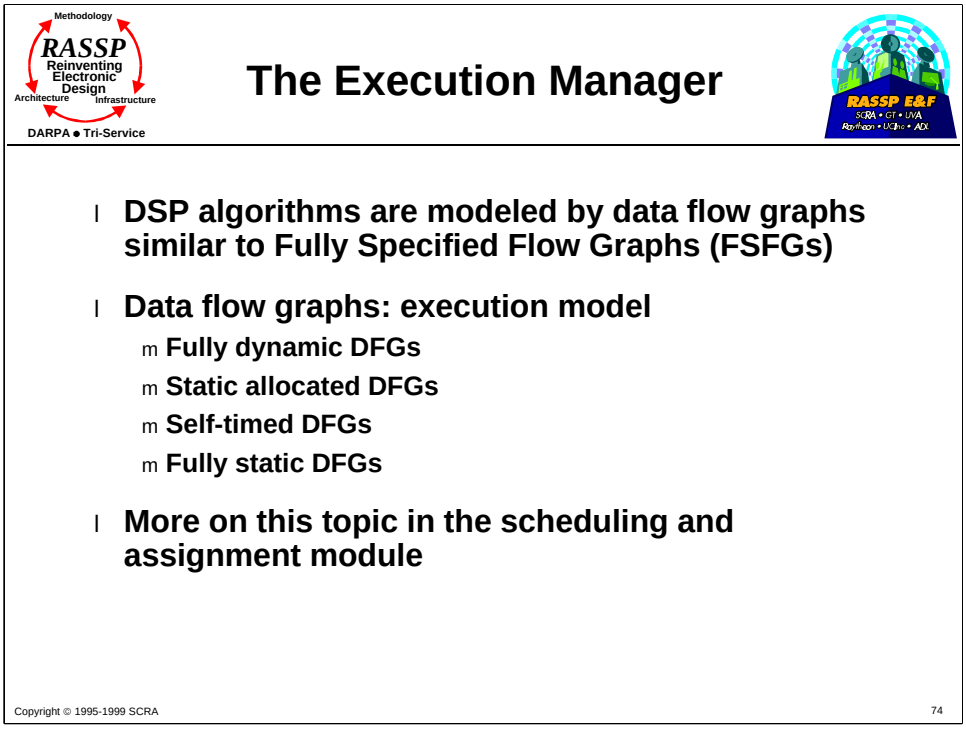
This environment was discussed earlier in the module.

Module Outline

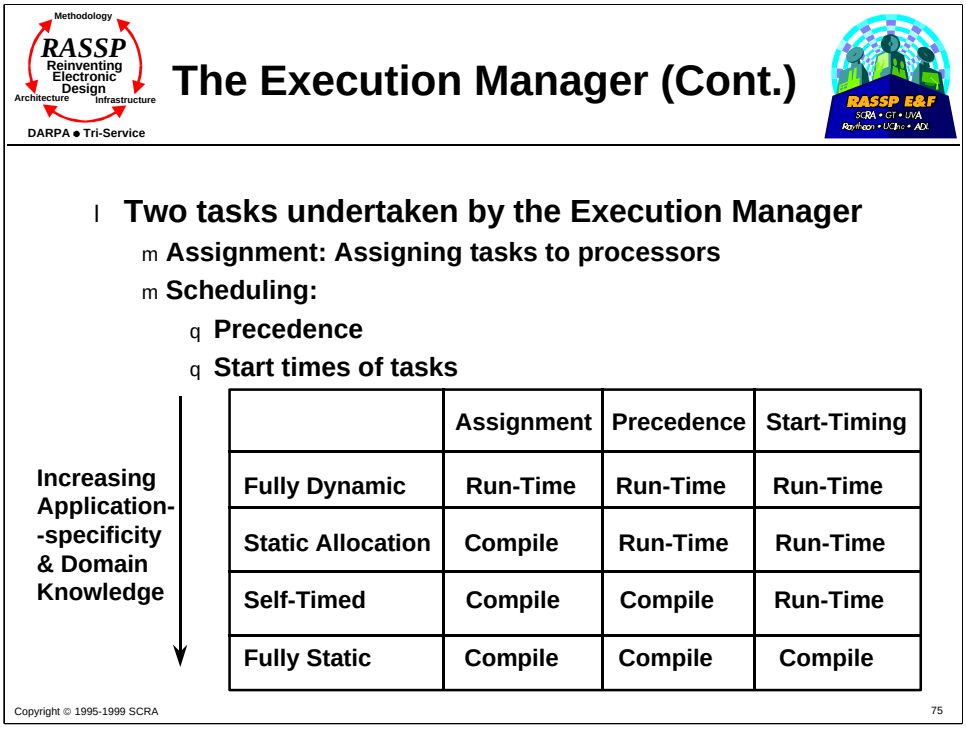
- Introduction
- Requirements Capture
- Fixed-Point Design - A RASSP Approach
- Simulation-Based Algorithm/Functional Design
- Network-Level DSP
- **Link-Level DSP**
- Signal Processing Simulators
- PGM/PGSE
- RASSP Software Generation
- Summary



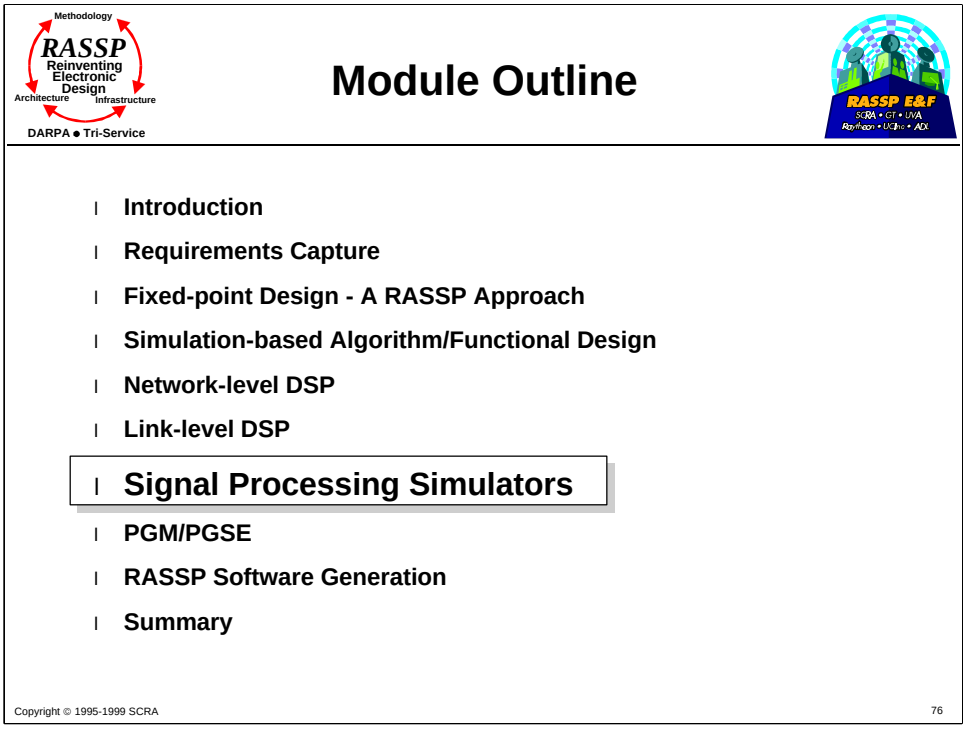
Link-level design environments.



The difference between the execution times of various simulators is closely dependent on the resource management issues related to the domain of computation.

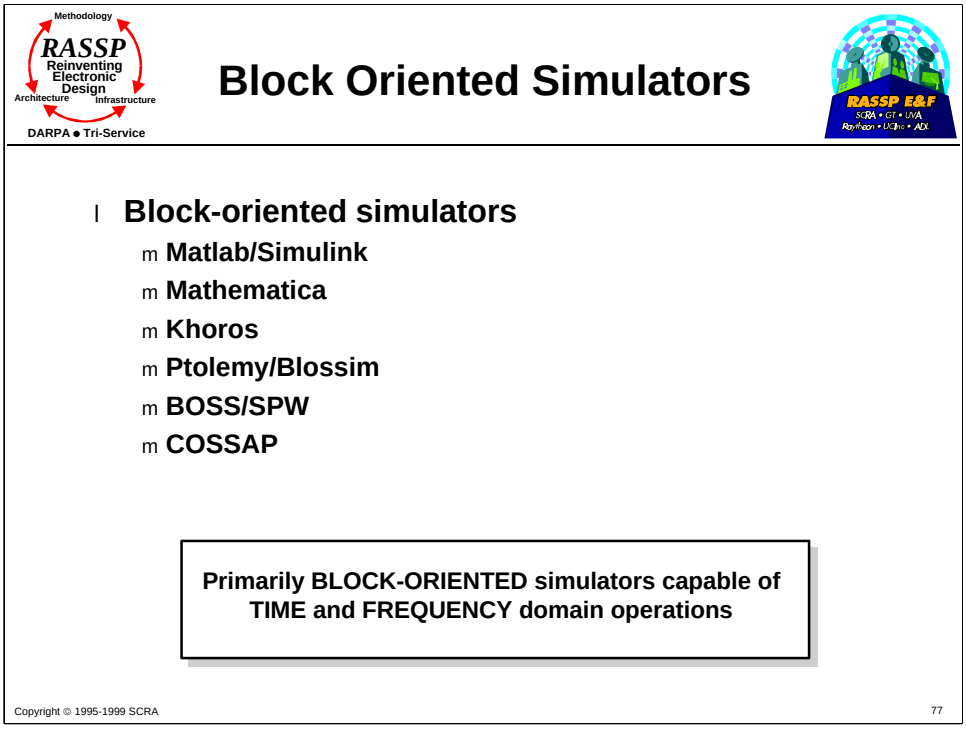


The impact of scheduling methodology, application specific of execution time.

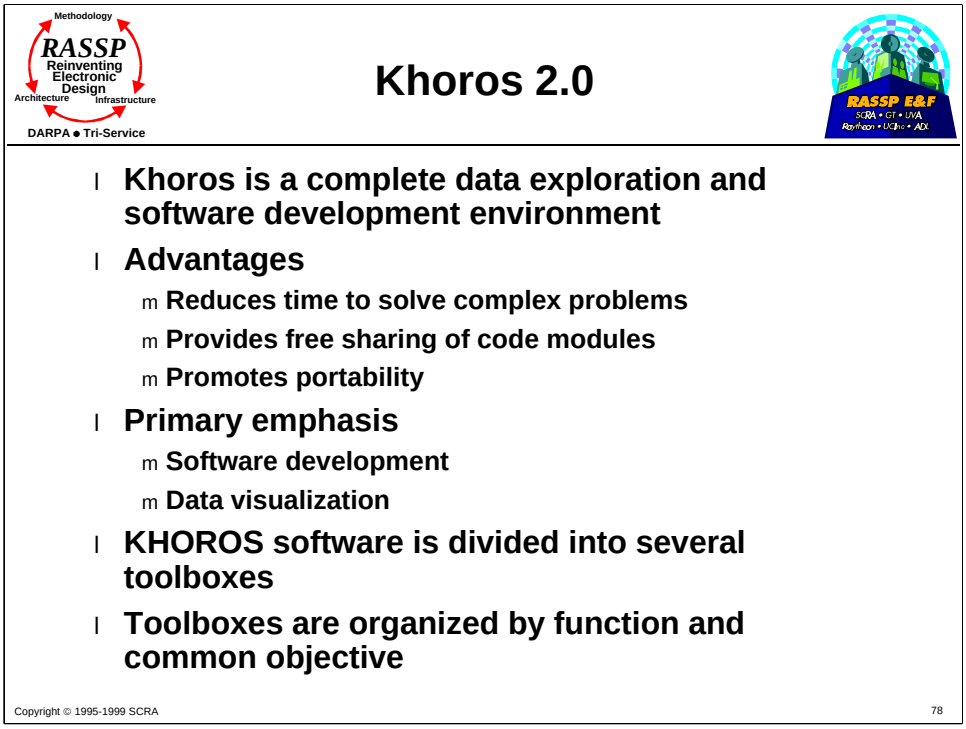


We will now describe some library-based signal processing simulators.

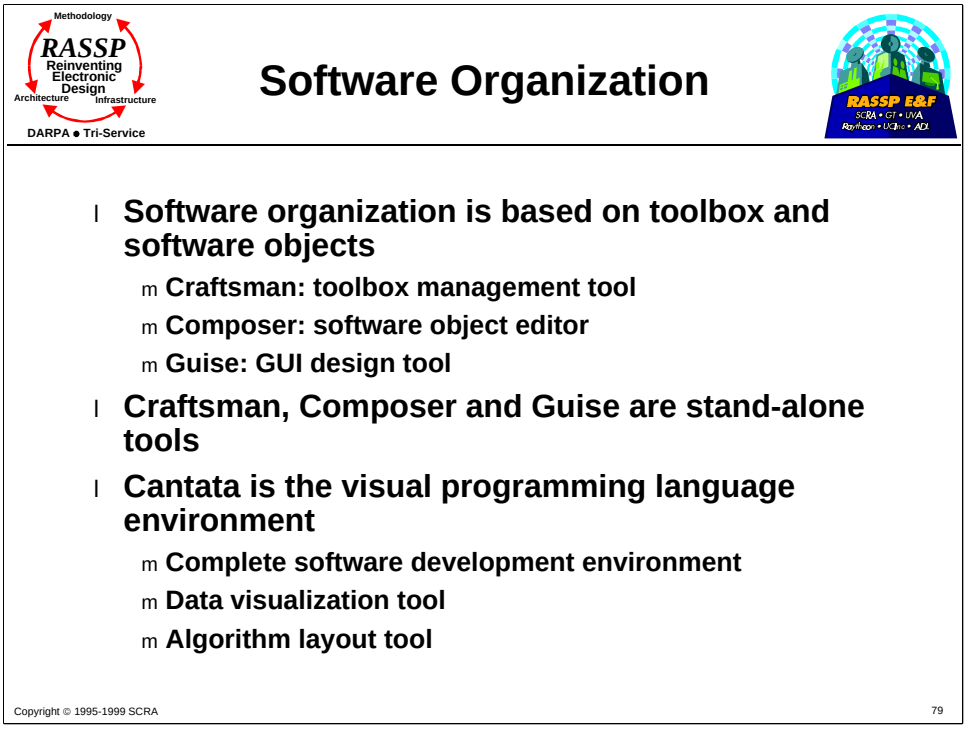
See http://ptolemy.eecs.berkeley.edu for a comprehensive list of publications related to the Ptolemy environment that is not repeated here.



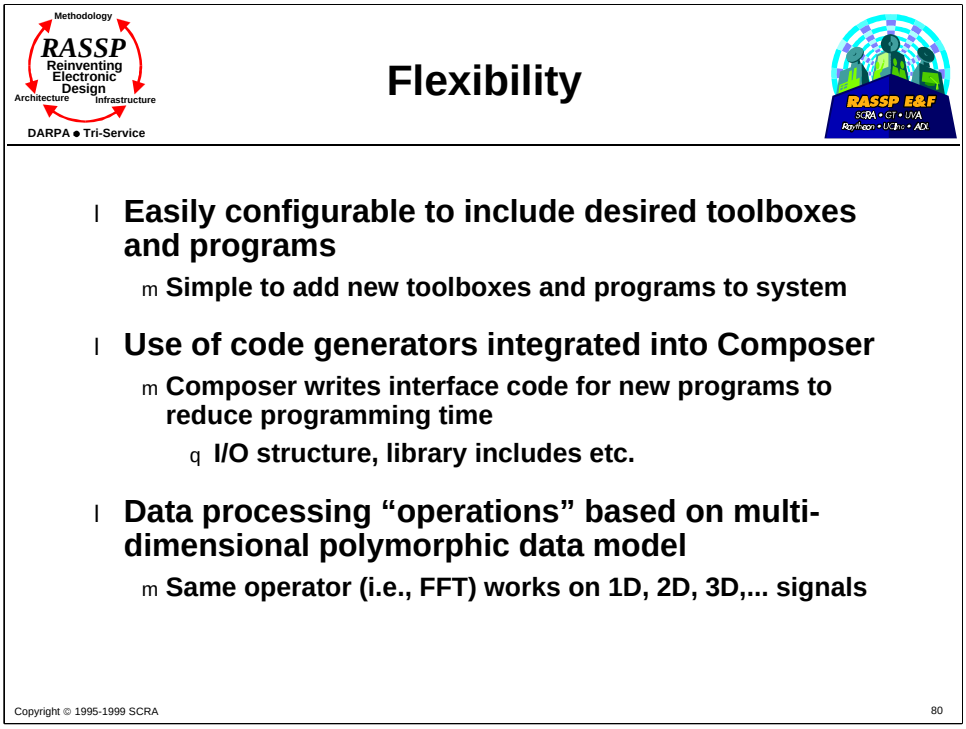
Signal processing environments relate to the lowest level of the hierarchy - the functional level.



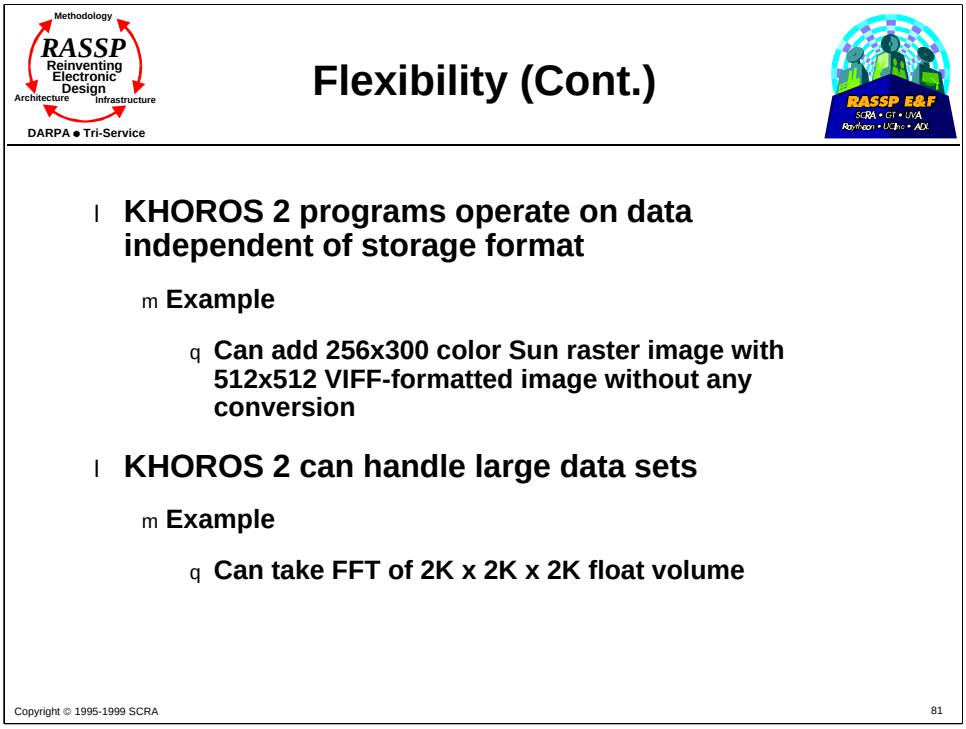
Khoros is widely used in image processing applications.



Additional information on Khoros may be obtained at http://www.khoral.com



Additional information can be obtained at http://www.khoral.com

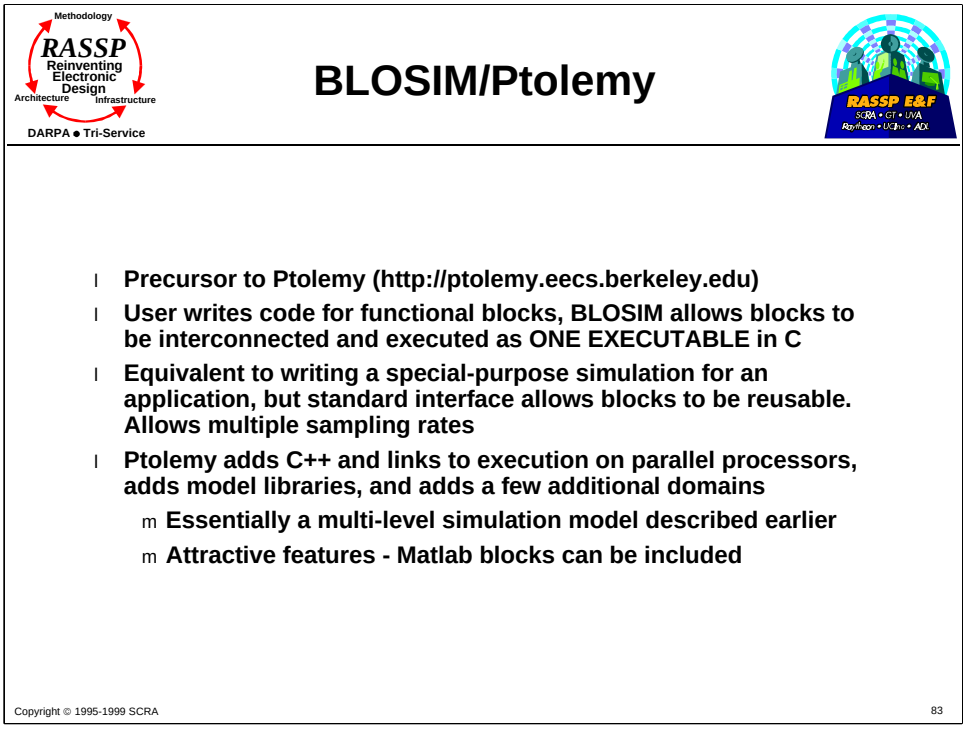


Additional information can be obtained at http://www.khoral.com

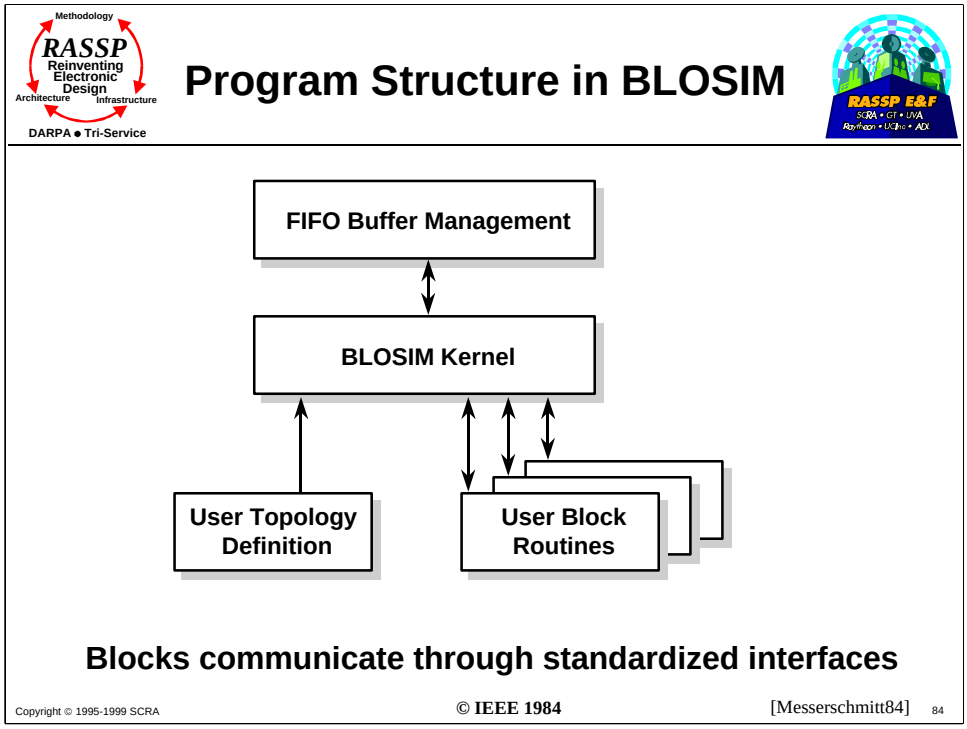
KHOROS Availability

- **Free access to KHOROS system**
  - **Not public domain, licensing is done through Khoros Research Inc.**
- **Available via FTP**
  - **USA ftp.khoros.unm.edu /pub/khoros/khoros2.0**
  - **USA ftp.sdsc.edu /pub/other/khoros/khoros-2**
  - **Germany ftp.uni-koeln.de /graph/khoros2.0**
- **Web site home page**
  - **http://www.khoros.unm.edu**

Additional information can be obtained at http://www.khoral.com

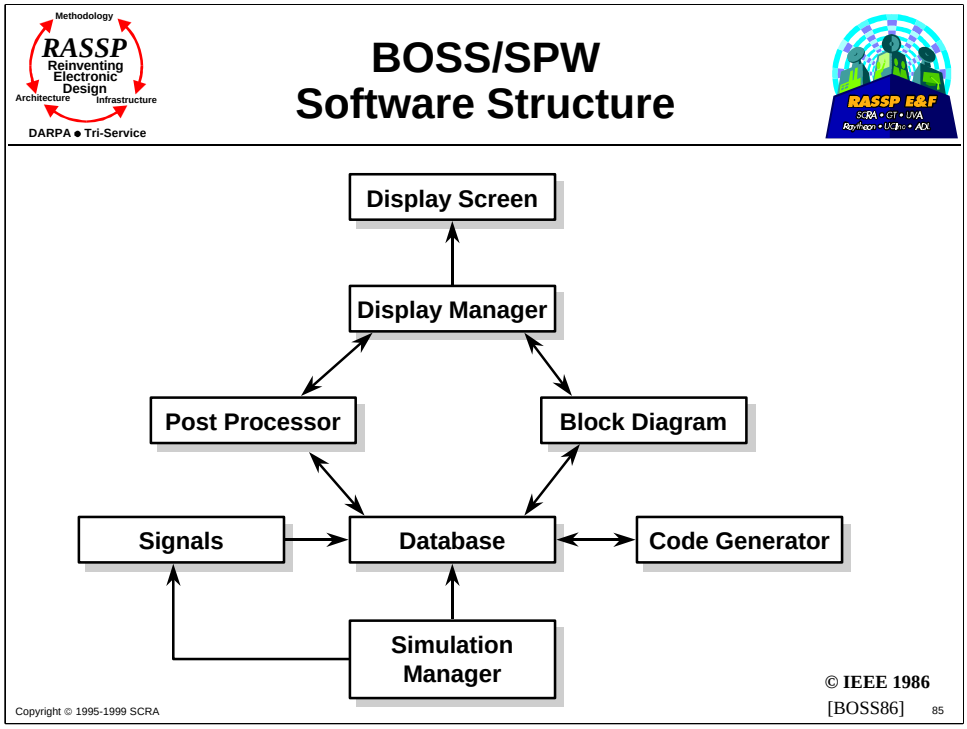


Blosim allows people to write efficient custom simulation programs yet reuse them later.

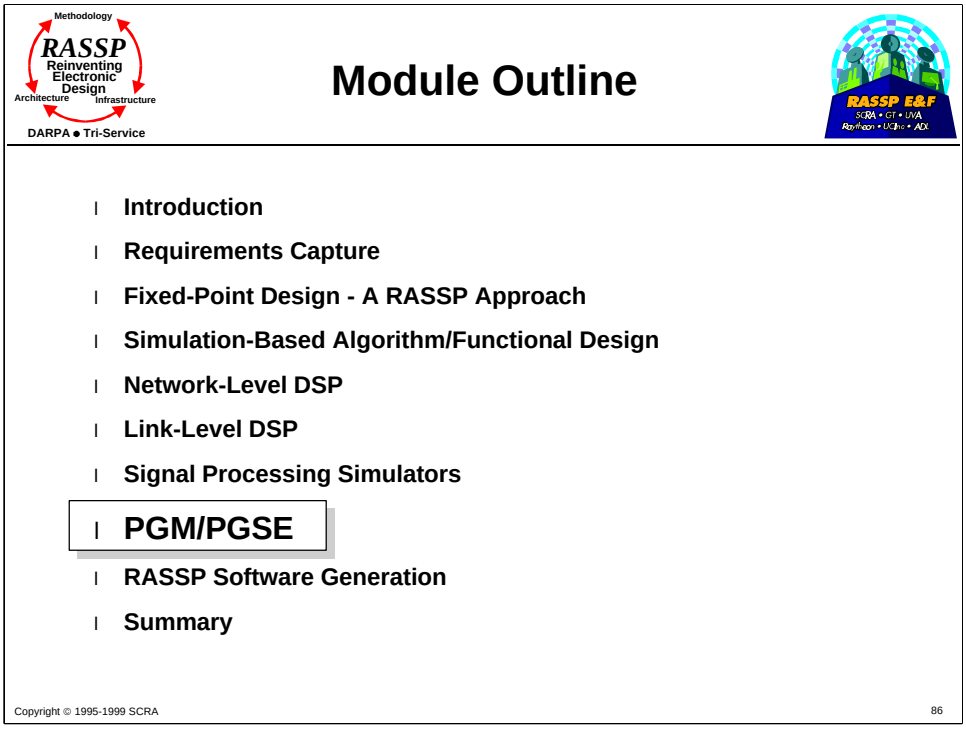


The user has to write all code. Blosim combines the blocks. There is extensive literature available on Ptolemy at

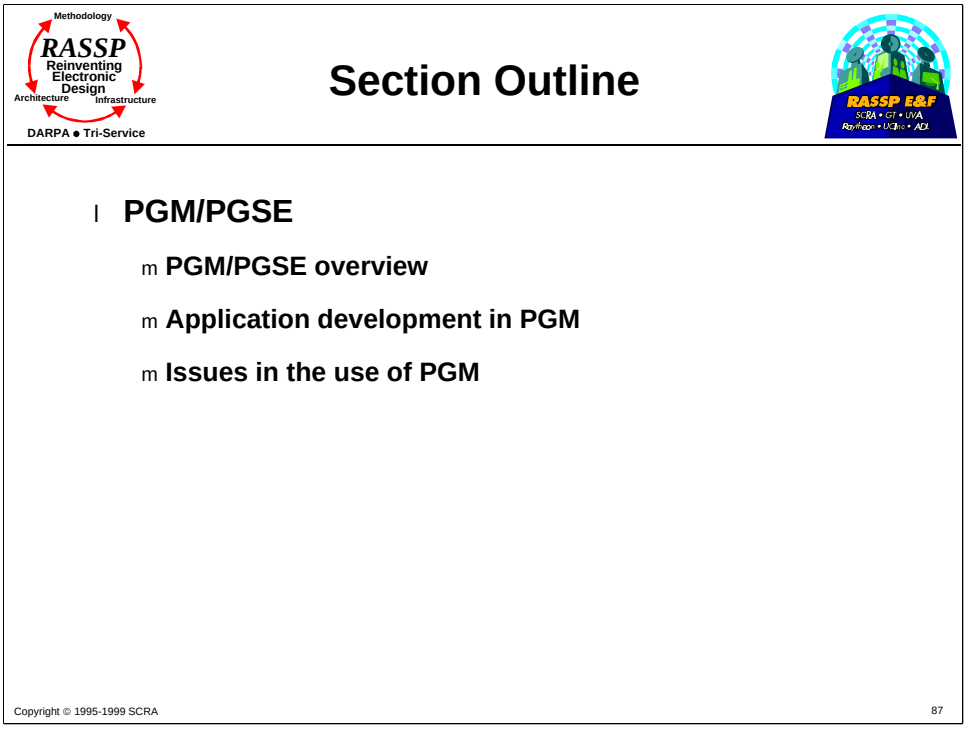
http://ptolemy.eecs.berkeley.edu regarding its recent progress and updates.



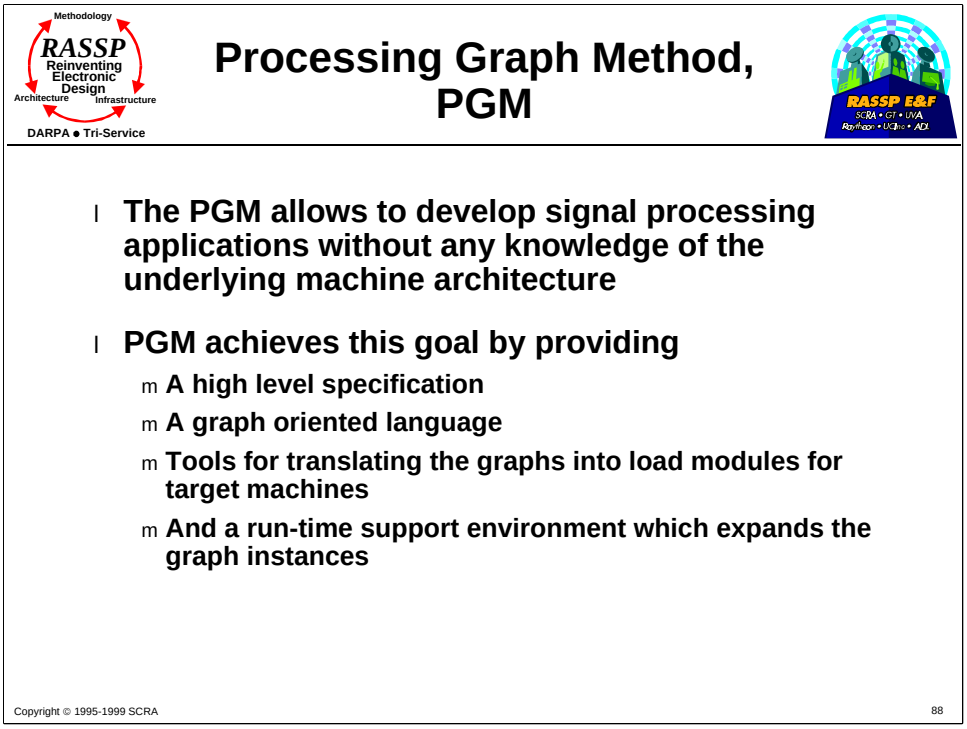
Another example (in addition to Khoros, Ptolemy, Blosim) is BOSS/SPW by Comdisco.



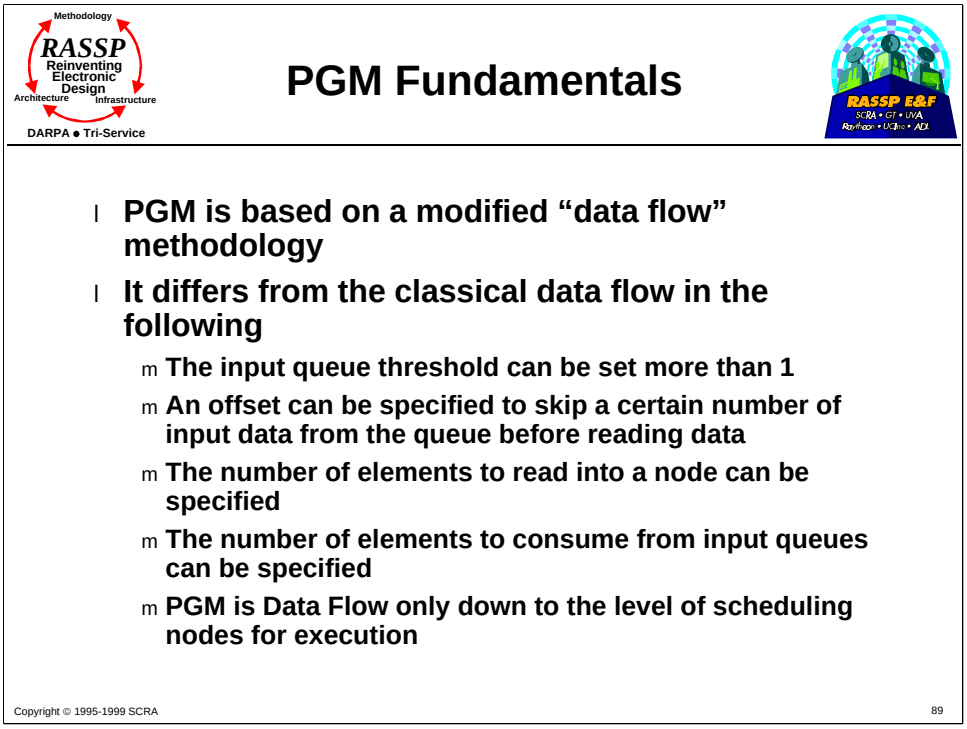
Processing Graph Methodology (PGM) is one of the original environments, developed by the US Navy, that is targeted towards RASSP applications (data flow and control flow specifications).



We now introduce PGM and PGSE environments with examples.

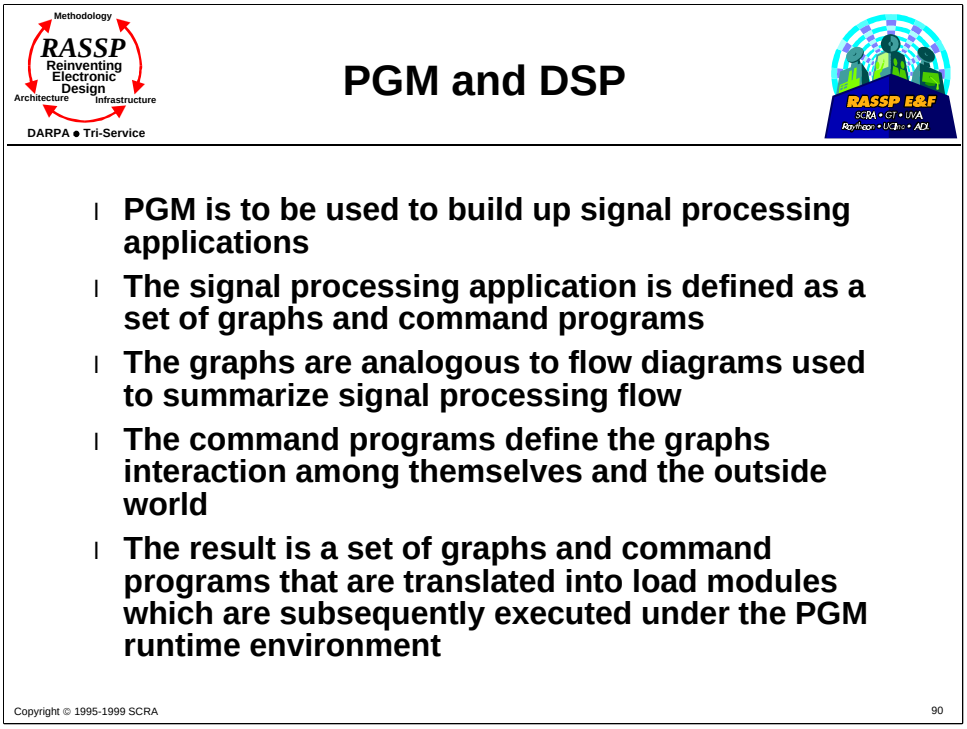


PGM facilitates an executable specification of the application.



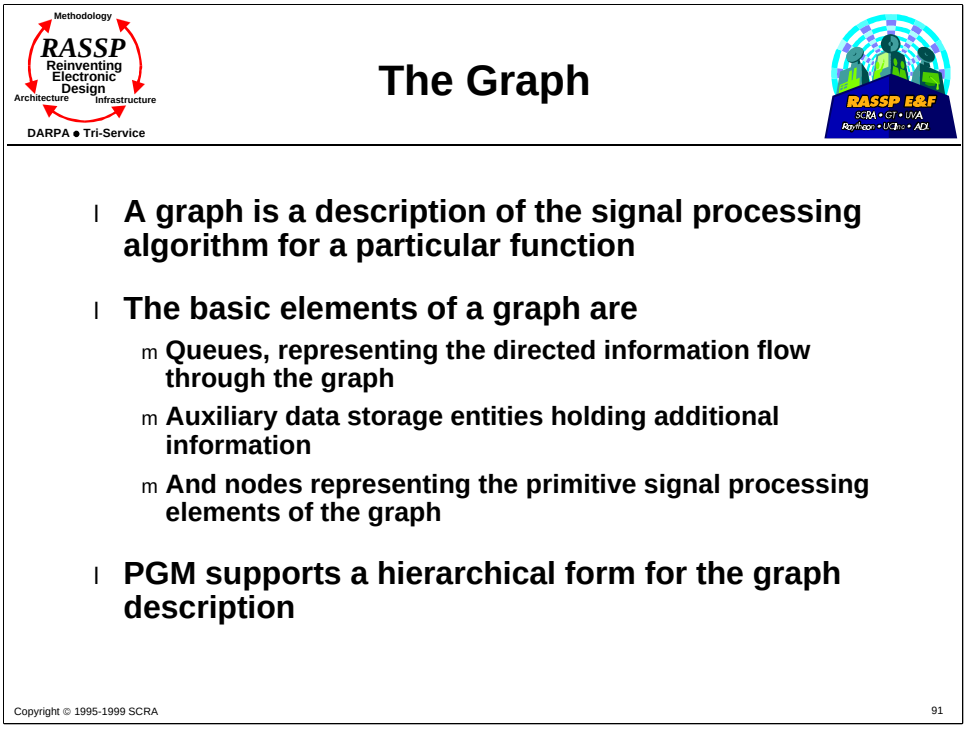
Dataflow Vs. Control flow

In a Control flow paradigm the order of execution of elements of the program are embedded in the program description. On the other hand, in a Dataflow paradigm the execution of the elements are based on the availability of the data. The environment provides a set of atomic operations that can consume data and produce data. Upon the execution of the program, the elements that have data ready will produce data that will enable other elements to be ready for execution.



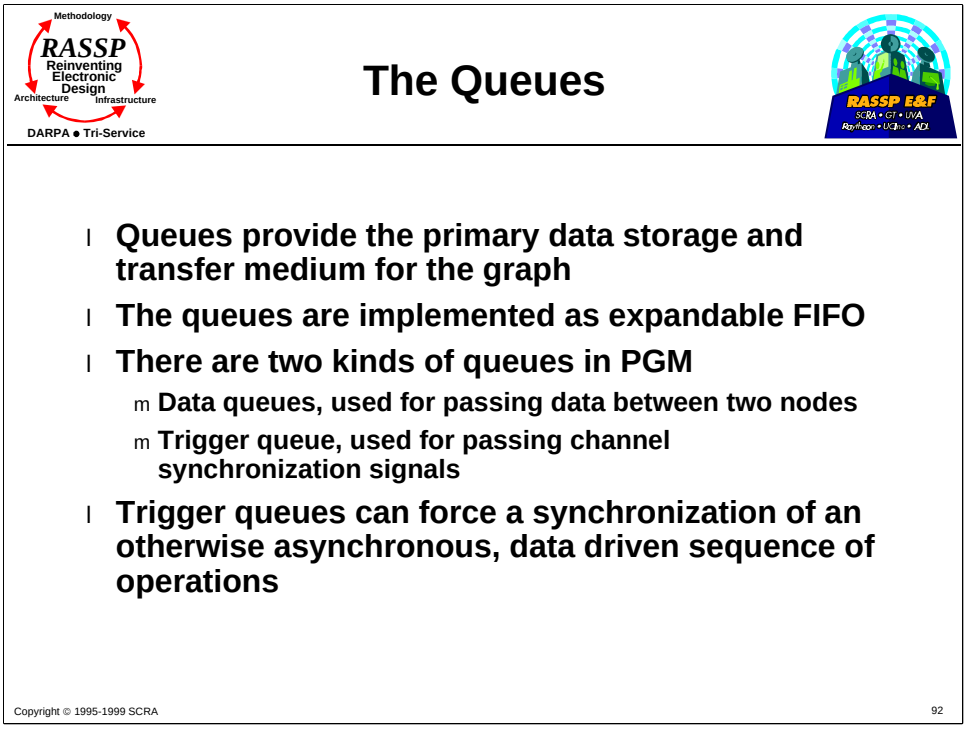
Description:

Signal processing applications are normally described using block diagrams that lend themselves very naturally to Dataflow paradigm. The block diagram is built out of certain black boxes that perform certain functionalities. These black boxes are further described based on more primitive signal processing elements. Each box in the block diagram processes the data that appears at its inputs and provides the result at the output where it is used by another box.



Description:

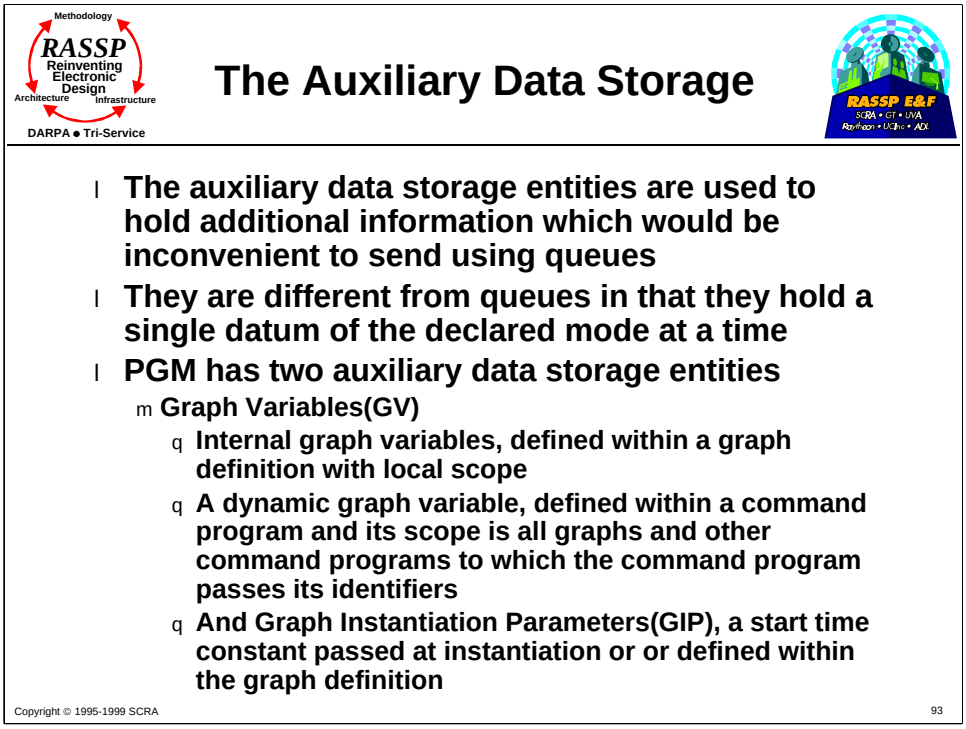
The basic elements of a Dataflow graph are graph nodes that process the data and arcs which guide the data through the nodes of the graph. The same basic ideas are true for PGM. The queues act as the arcs of the graph, where the head of the queue corresponds to the arrow of the arc, and the primitive signal processing elements provide the atomic operation of the graph.



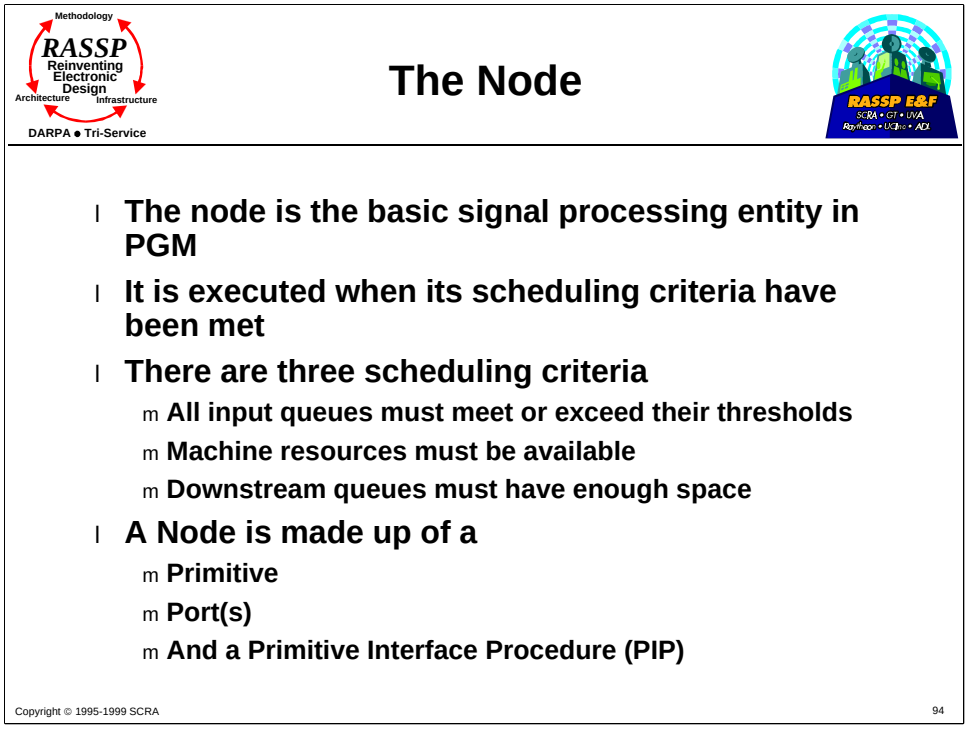
Description:

FIFO are memory storage devices that are accessed sequentially based on their arrival order to the device. The first element to arrive is the first element to be supplied by the first read to the device.

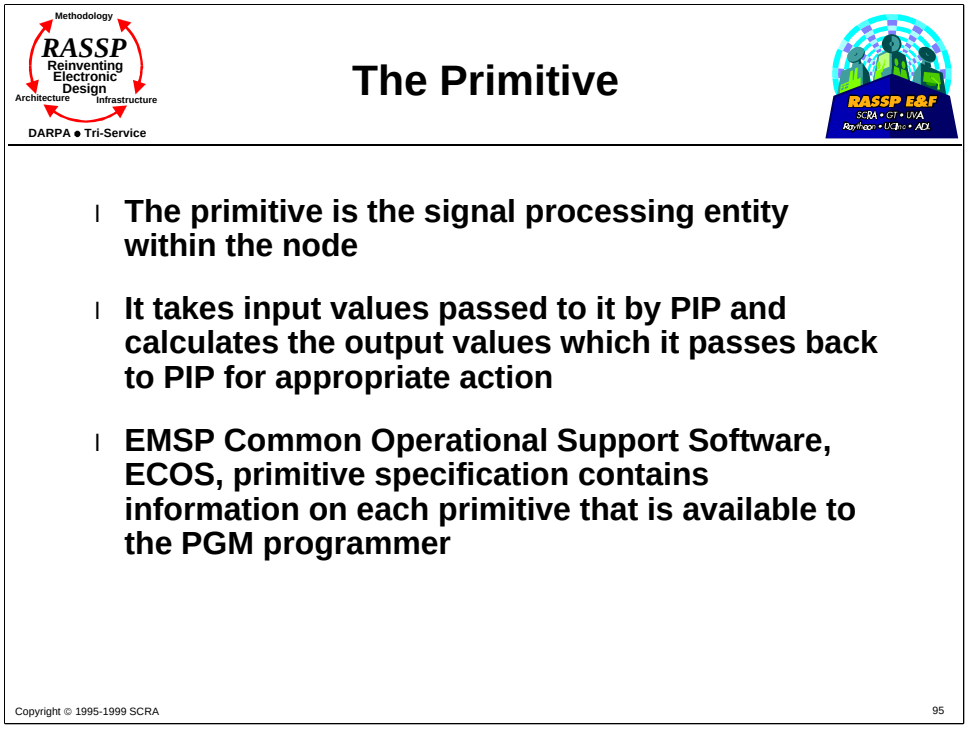
Trigger queues are used to send trigger pulses to graph nodes so that the execution of a number of them can be synchronized.



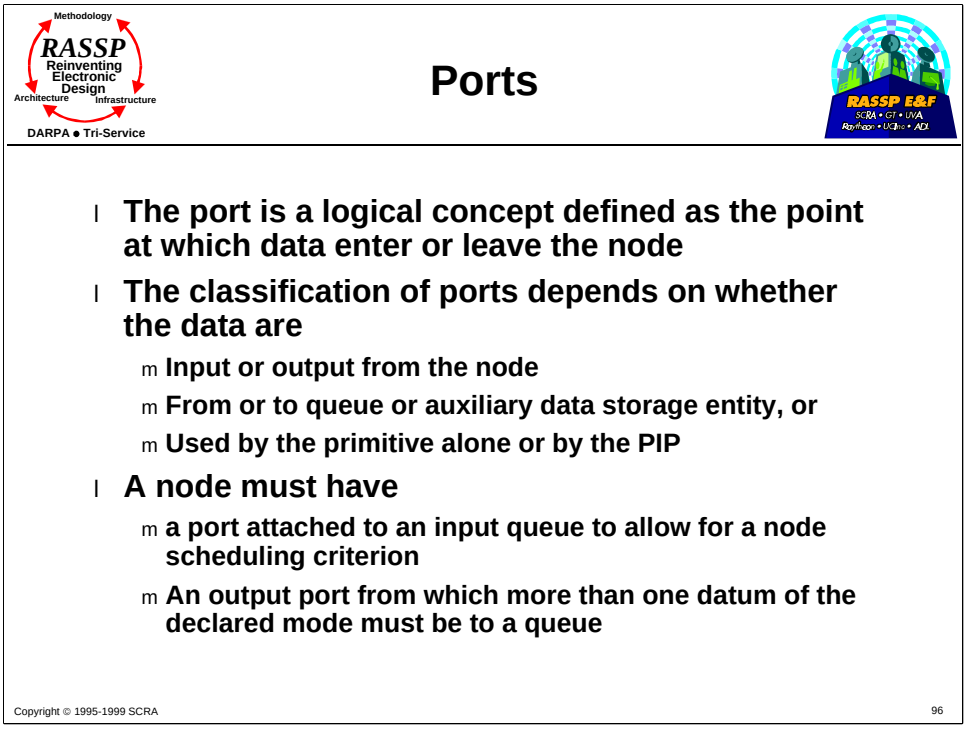
Additional data structures used in PGM are described.



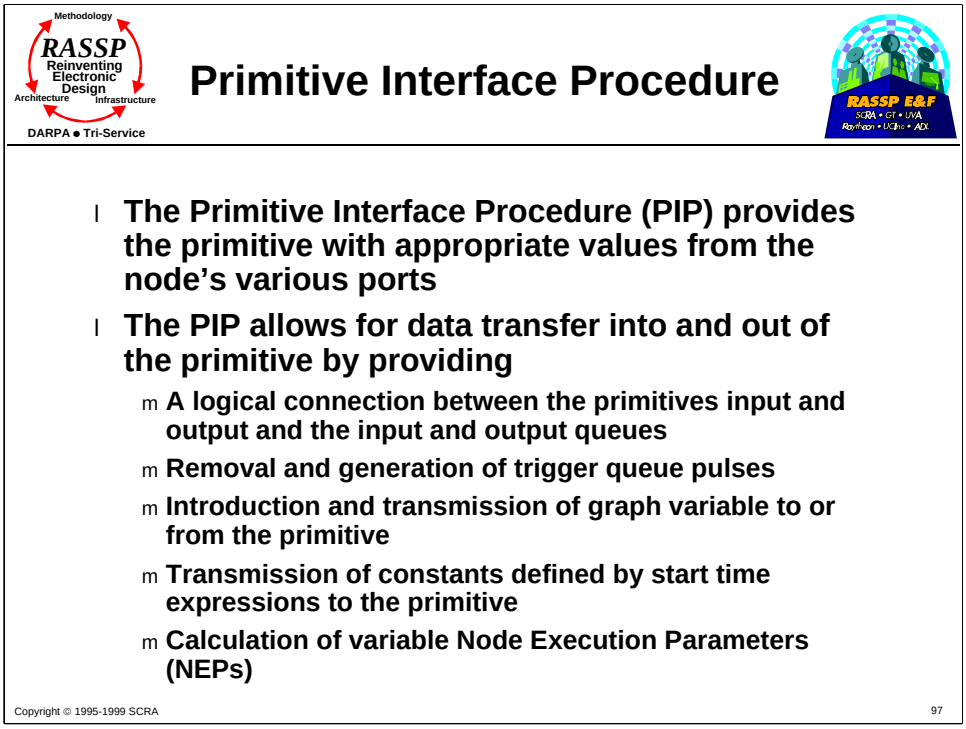
Nodes store the computational primitives and their interfaces.



The primitive is the mathematical operation.



Ports are part of the node interface



Description:

It is important here to differentiate between port input/output and PIP.

PRIM_IN Vs. PIP_IN:

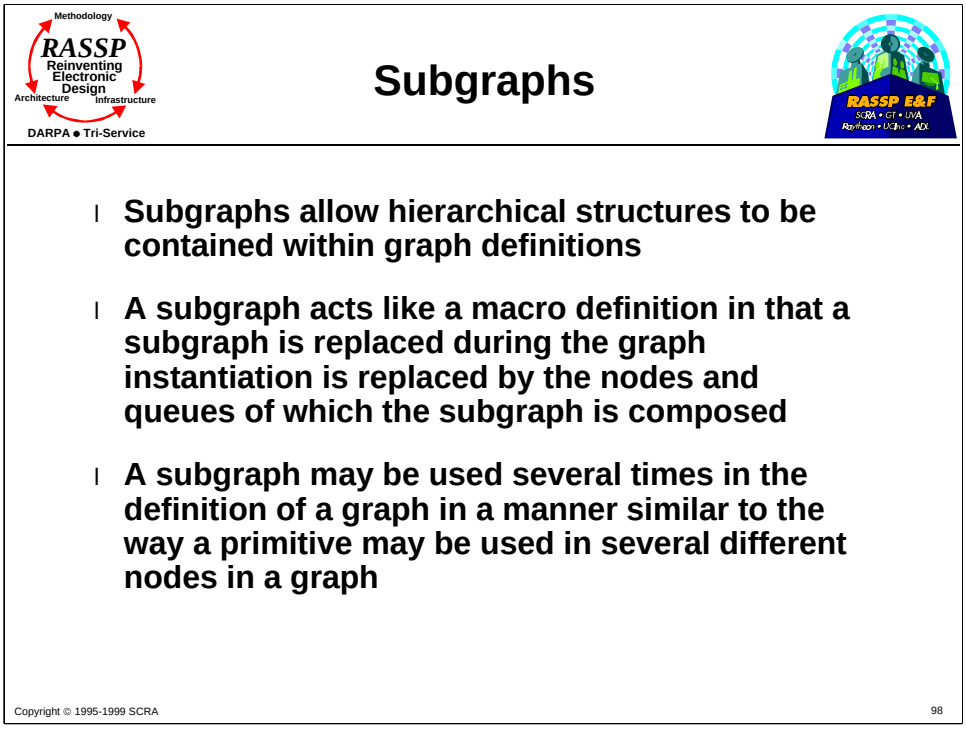
Neither start time or run time expressions can not be attached to a PIP_IN port. Further restrictions also apply:

1) A queue must be of mode integer or trigger.

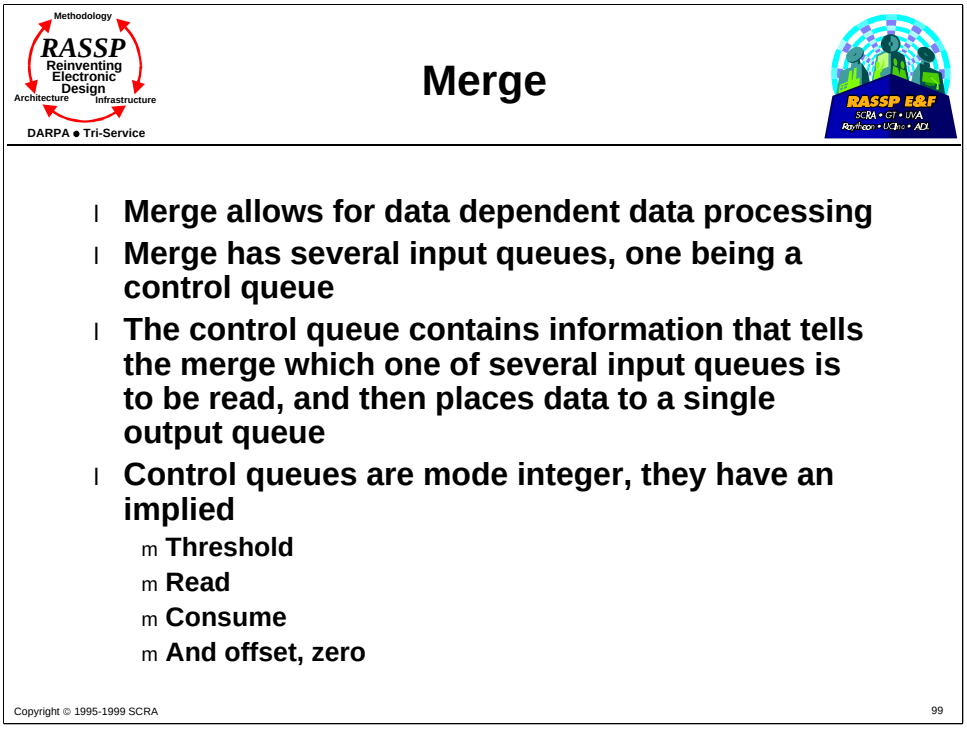
2) If the queue is of mode integer, it can only be used for, run-time expression in a PRIM_IN, selector, or variable NEP.

PRIM_OUT Vs PIP_OUT:

Only queues may be connected to a PIP_OUT port, queues of mode integer or trigger.

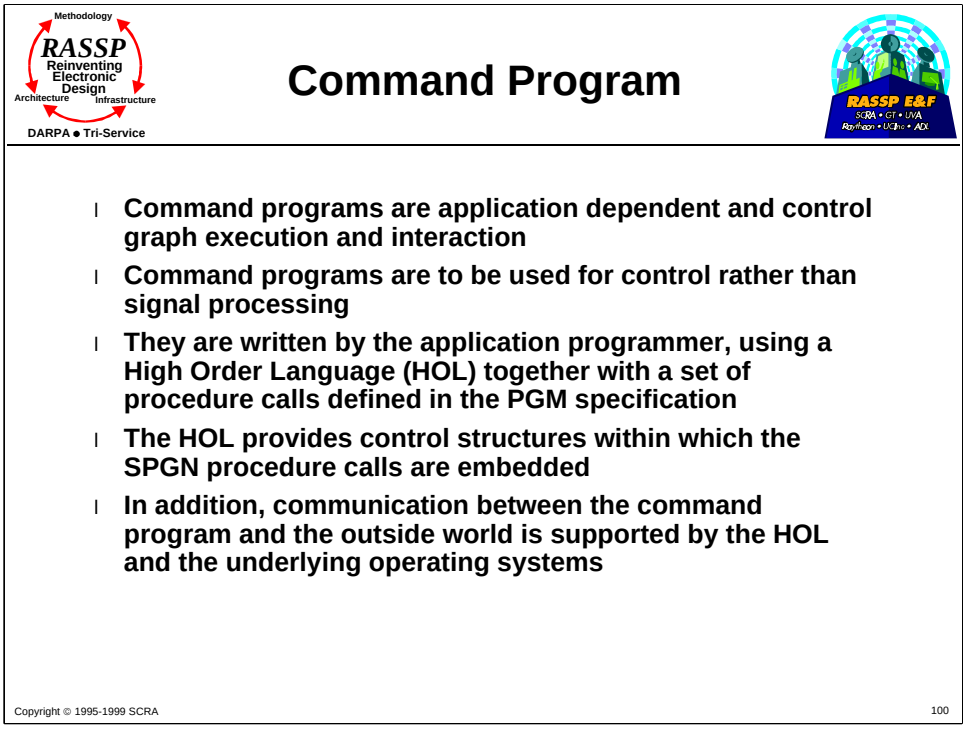


Subgraphs allow the construction of super-nodes (hierarchy)

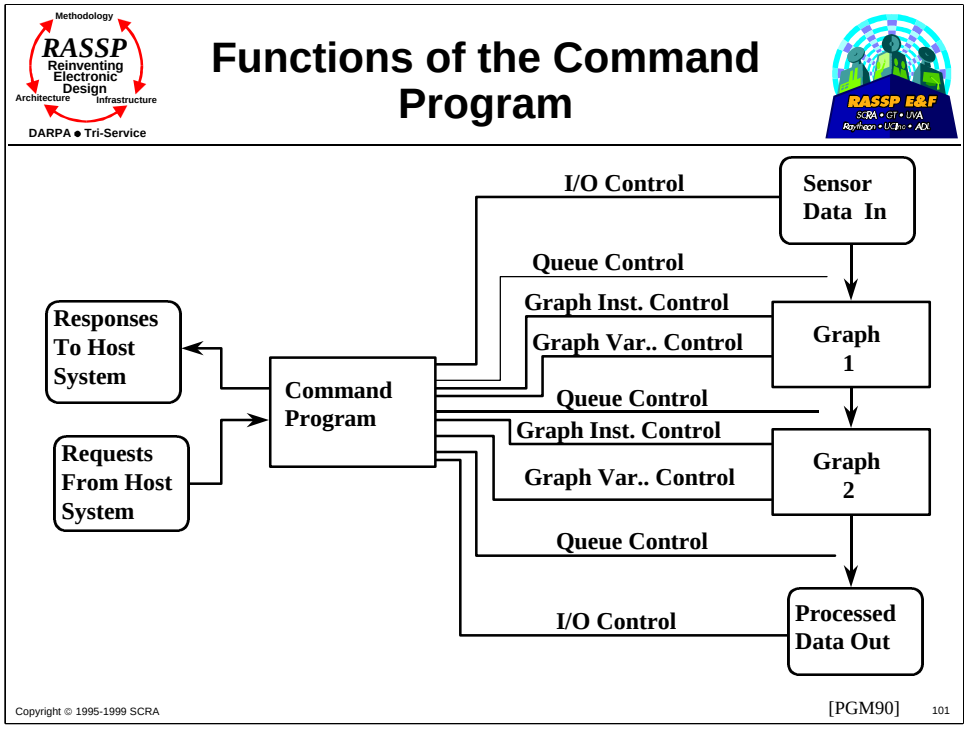


Description:

The implied threshold, read, consume, and offset values are set by PGM and can not be changed by the graph-writer.

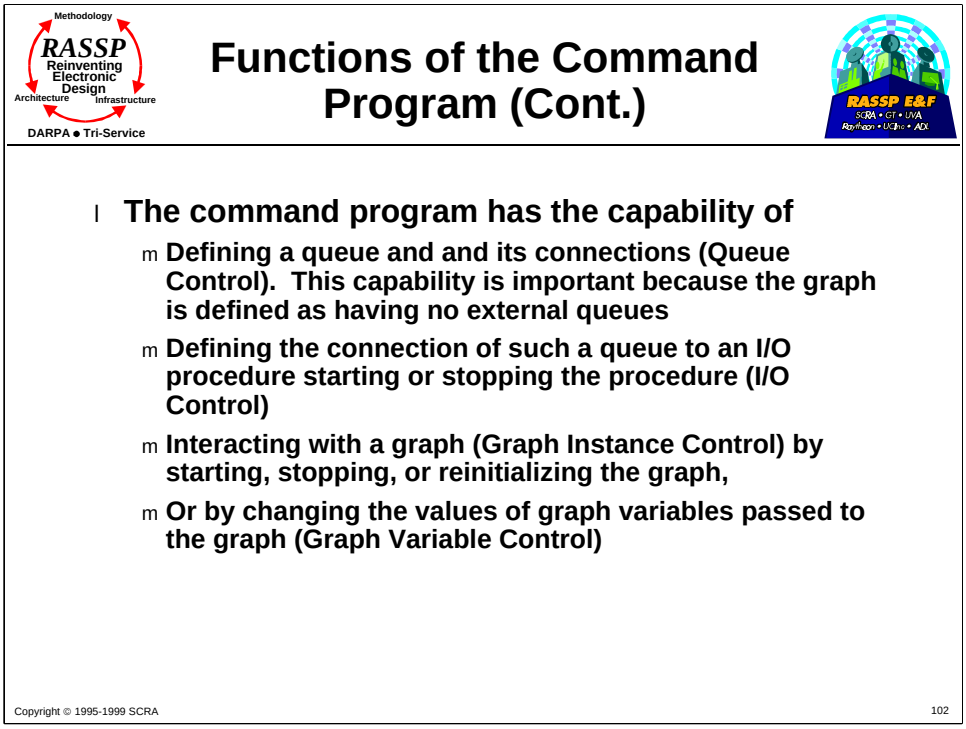


The command programs control the execution of the signal task graphs.



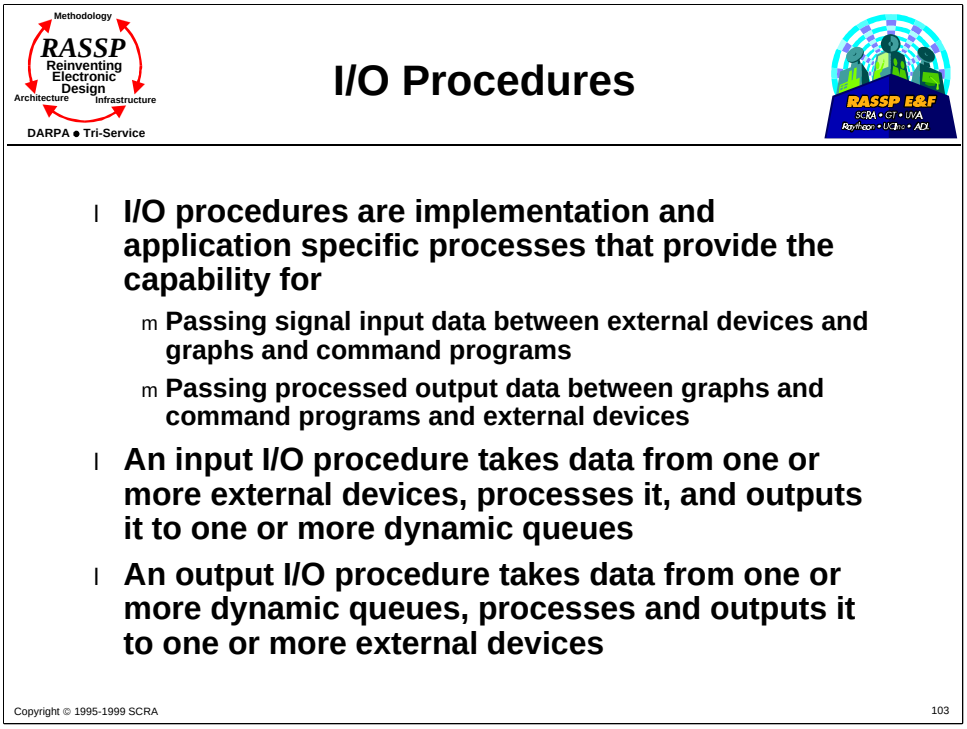
Description:

The figure displays the functions of the command program. Notice that at one end it interfaces to the host system and at the other end it interfaces to graphs and dynamic queues that it creates and manages.



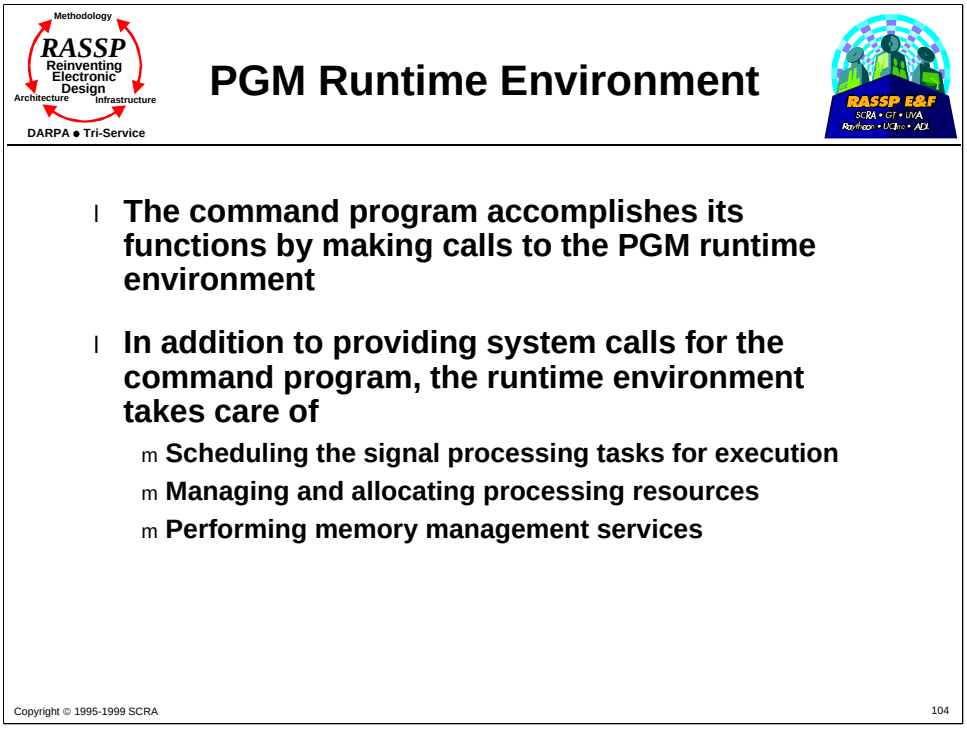
For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm



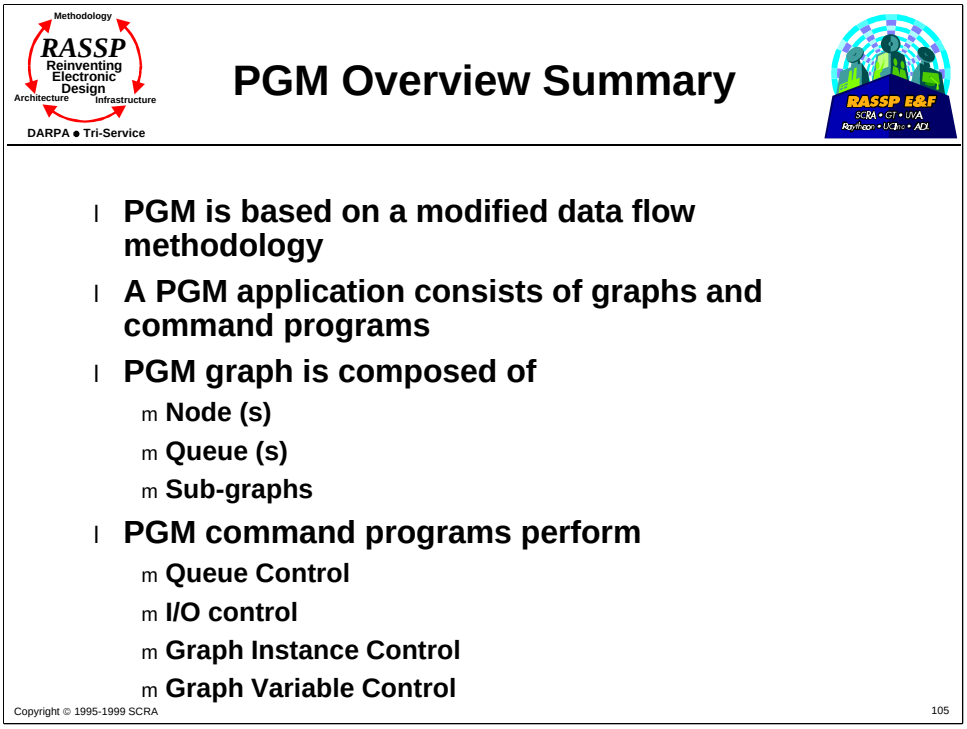
For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm



For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm

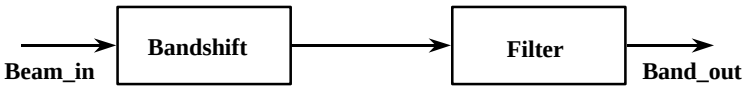


For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm

Application Development in PGM

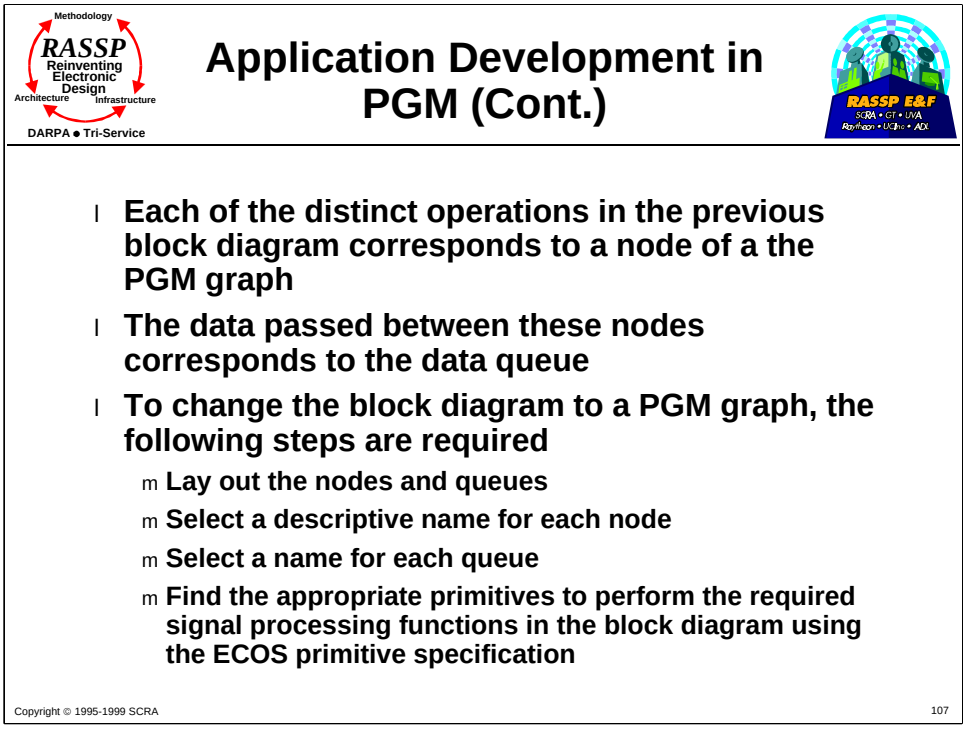
- The starting point for developing a PGM graph is a signal processing flow diagram
- Consider the simple example band definition filtering on one beam of acoustic time-series data
- The frequency band is defined by the bandshift and subsequent low pass filtering. The sampling rate is to be reduced by a factor of 8 (8:1 decimation)



Signal Processing Flow Diagram for Band Definition Filtering

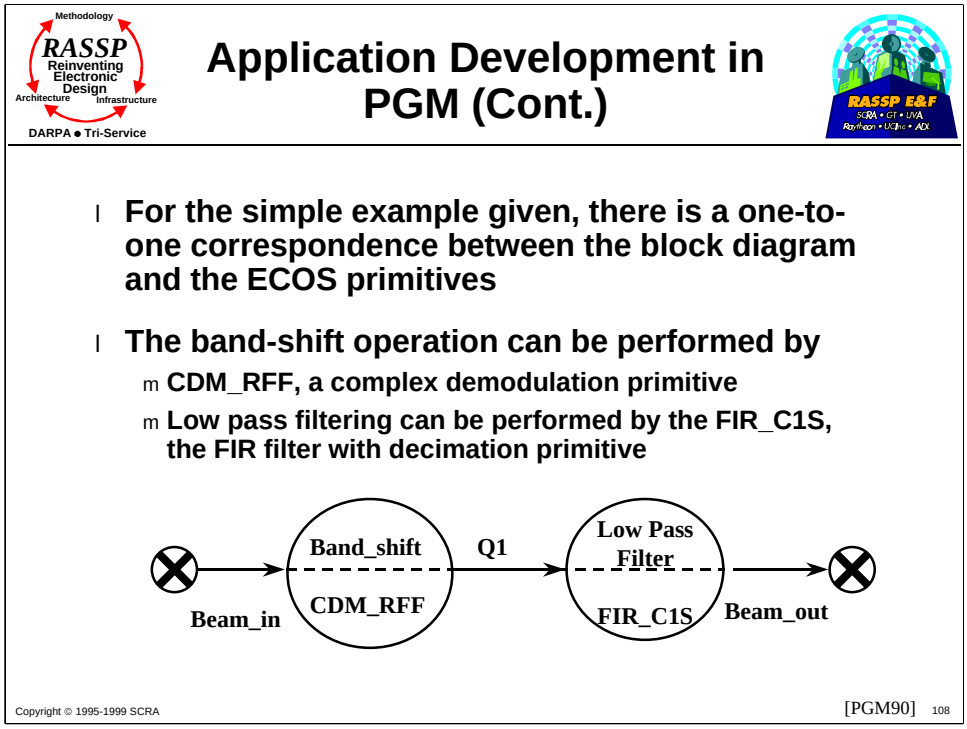
For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm



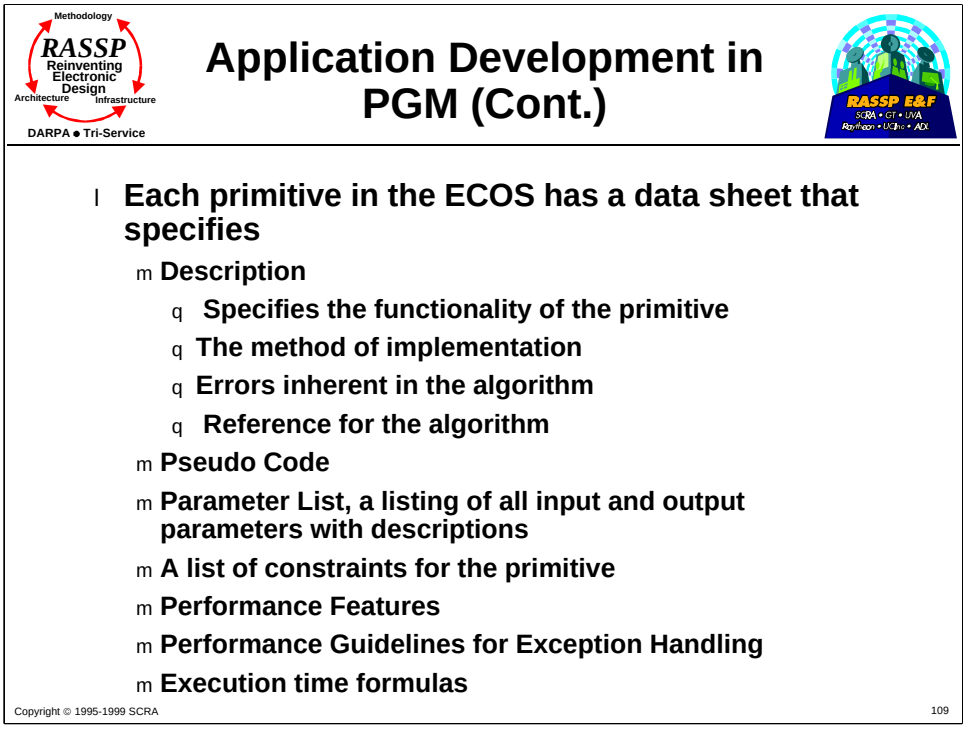
Descriptions:

The basic steps of converting a signal flow graph to a PGM graph is presented.



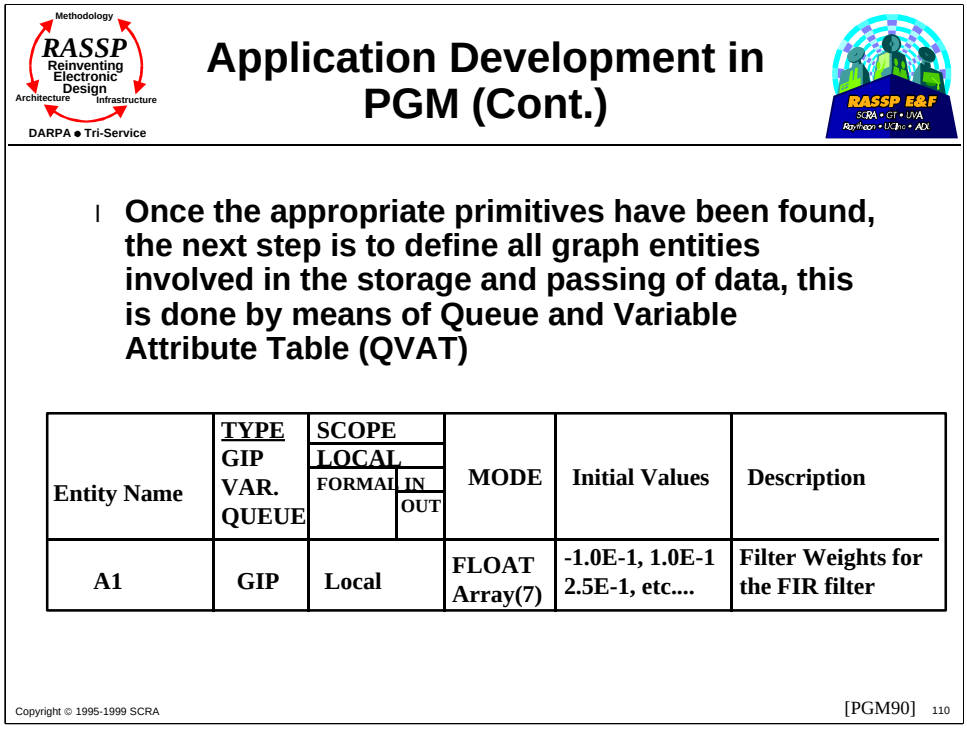
Description:

The complete description of these primitives are available in Q003 specification library.



For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm



Description:

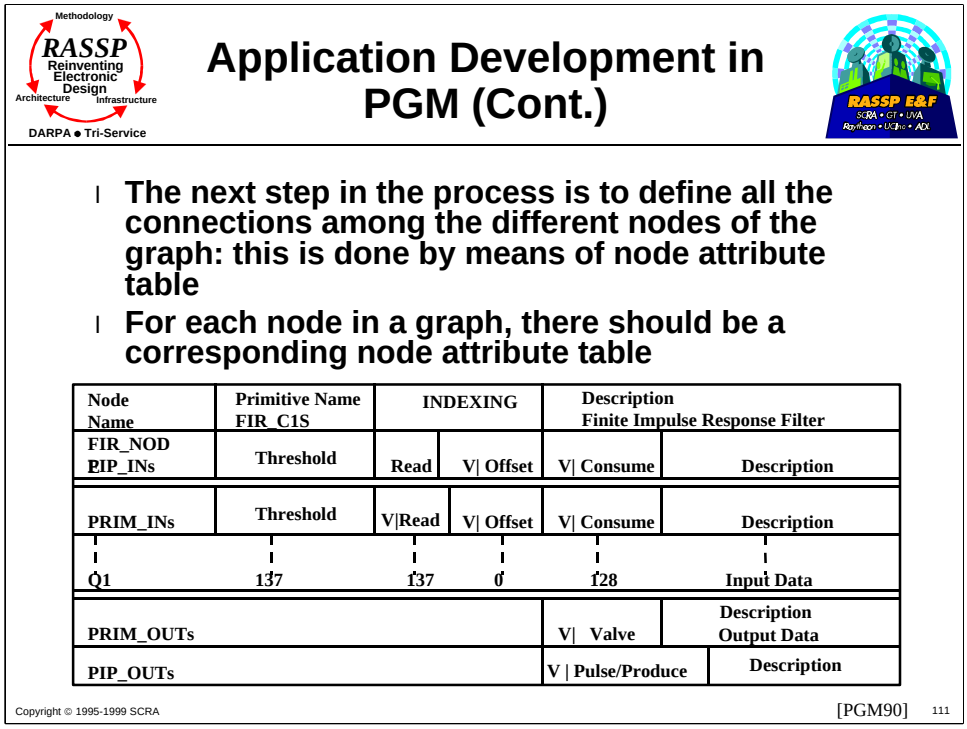
Entity Name refers to the data storage element.

Type specifies one of GIP, QUEUE, or VAR...

Scope specifies the degree to which the entity is visible to others.

Mode defines its data mode.

Initial values contain the start up values for the entity.



Description:

The header contains information about the node.

PIP_IN, PRIM_IN, PRIM_OUT and PIP_OUT all specify information with regard to who they are connected and the threshold, read, offset, and consume amounts for each queue.

Application Development in PGM (Cont.)

- **Functions of command program fall into five categories**
  - **Command program control**
  - **Input/Output control**
  - **Graph instance control**
  - **Queue control**
  - **Graph variable control**
- **A combination of HOL and command program, SPGN, procedures are used to**
  - **Define the control flow structure**
  - **I/O to an "operator" or controlling computer**
- **Command program, SPGN is used to coordinate the interactions**
  - **Between command programs and graph instances**
  - **Between two command programs**

For additional information please consult

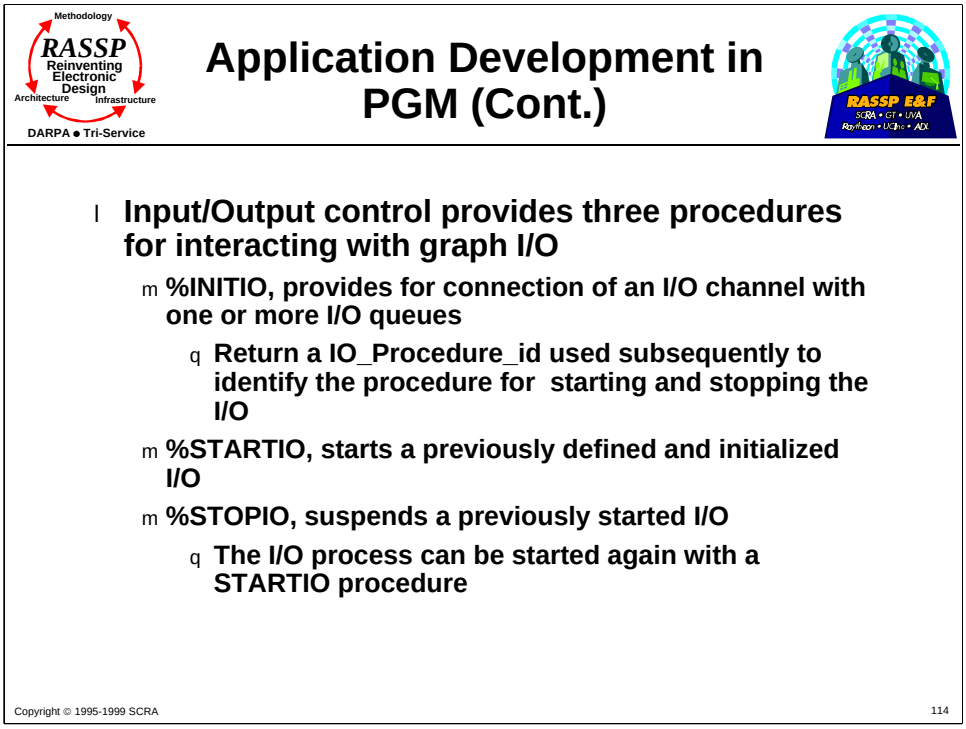
http://pms428.uswinfo.com/pgm-1.htm

Application Development in PGM (Cont.)

- **Command program control**
  - **One command program is designated as the initial command program and is started at startup**
  - **All other command programs are created (spawned) by this initial command program**
  - **There are two procedures for command program control**
    - **%SPAWN, used to start another command program and to pass arguments to it**
      - **The procedure returns the identifier Command_Program_id**
    - **%ABORT, used to abort a command program**
      - **A command program can only abort itself or any other command programs that it had spawned.**

For additional information please consult

http://pms428.uswinfo.com/pgm-1.htm

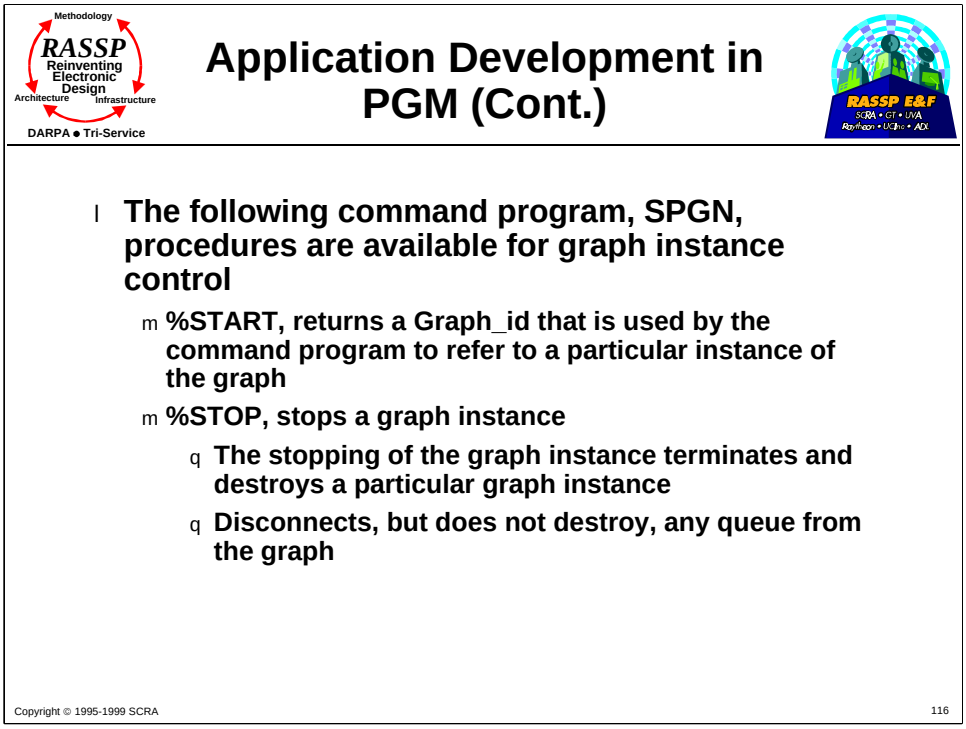


For additional information please also check the PGM page at http://www.ait.nrl.navy.mil/pgmt/

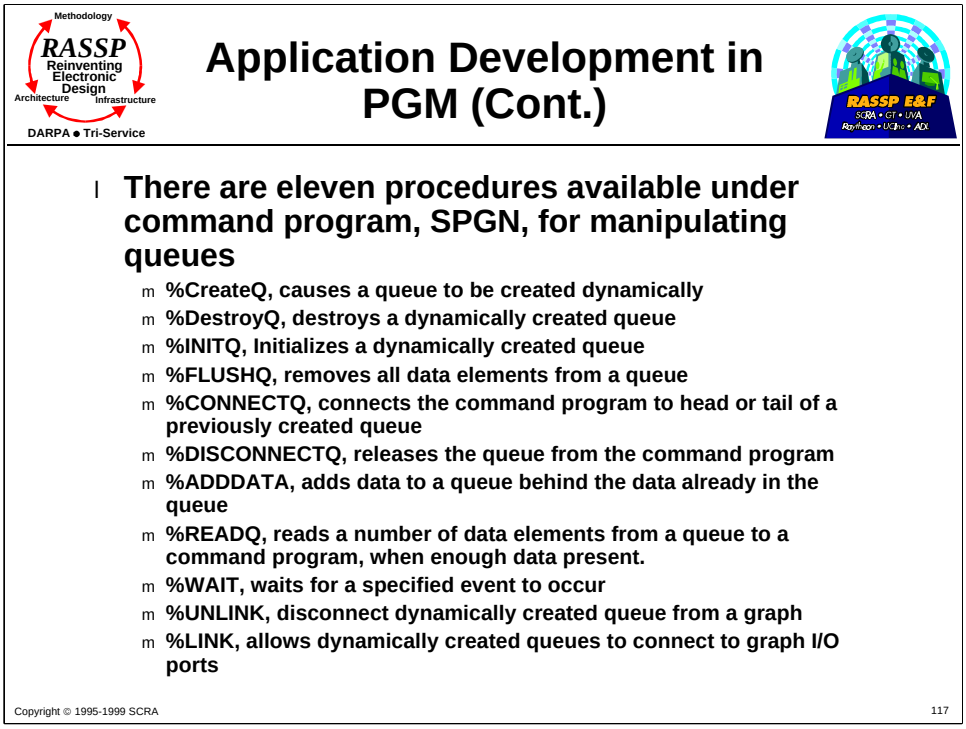
Application Development in PGM (Cont.)

- Graph instance control is responsible for transforming the applications graph into a static description of the signal processing graph
- In order for a graph to be executed, graph realization must be used to
  - Create a particular graph instance with particular
    - Queues
    - Nodes
    - Graph variables
- It is possible to have many different graph instances that started from the same graph realization active at the same time

See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.



See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.



See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

Application Development in PGM (Cont.)

- **PGM provides four procedures for command programs to interact with graph variables**
  - **%CREATEGV**
    - **Dynamically creates a graph variable and returns an identifier which it can be referenced**
  - **%DESTROYGV**
    - **Destroys a a previously dynamically created graph variable**
  - **%READGV**
    - **Enables the command program to read a data element or group of array elements from a graph variable**
  - **%WRITEGV**
    - **Used by the command program to place data onto a graph variable**

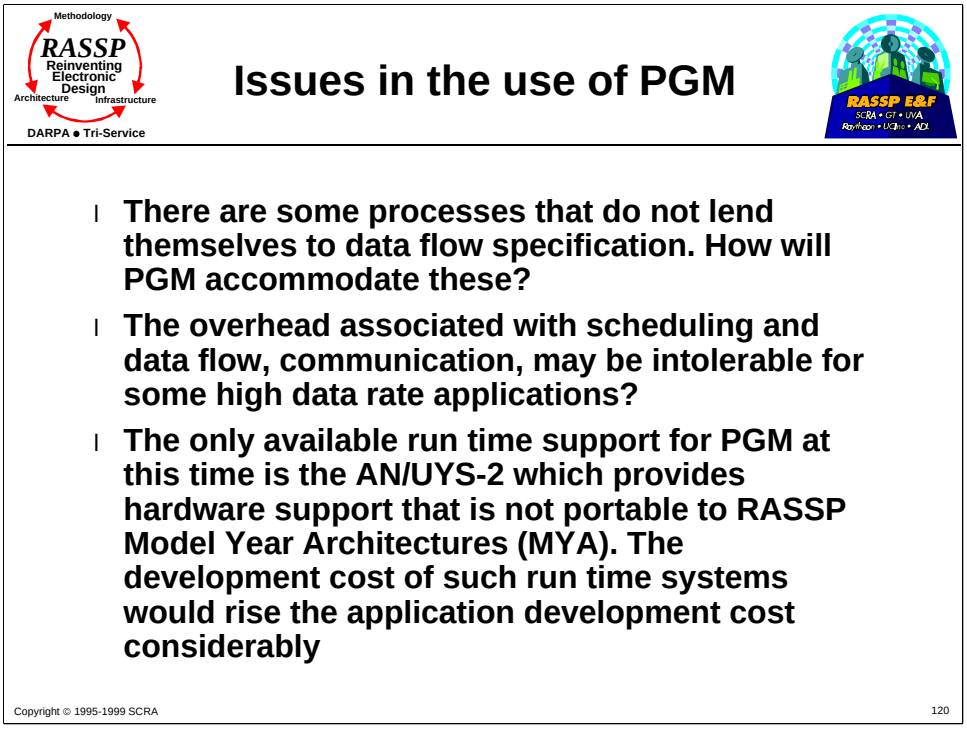
See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

Application of PGM in RASSP

- PGM may be used to capture the algorithmic/functional flows developed by higher level math tools
- The data flow graph functional specifications may be easily expanded to full specification of application by adding
  - Family structure to specify channelization
  - Fixed and variable parameters to govern the processing within each node of PGM graphs and relative node execution rates, and sequences of the nodes
  - Variations of data flow through paths of the graph
- The graphical programming has demonstrated a reduction in development cost by order of magnitude
- PGM inherently is capable of making all aspects of application execution visible.

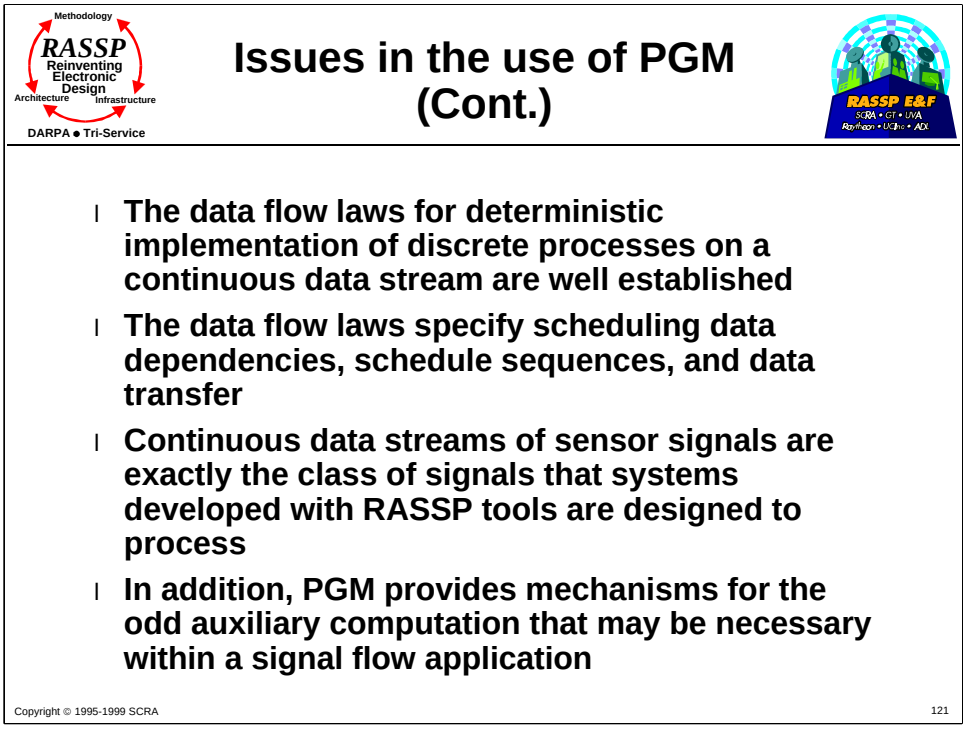
See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

[MMC/LMC 94]



See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

[MMC/LMC 94]



See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

[MMC/LMC 94]

Issues in the use of PGM (Cont.)

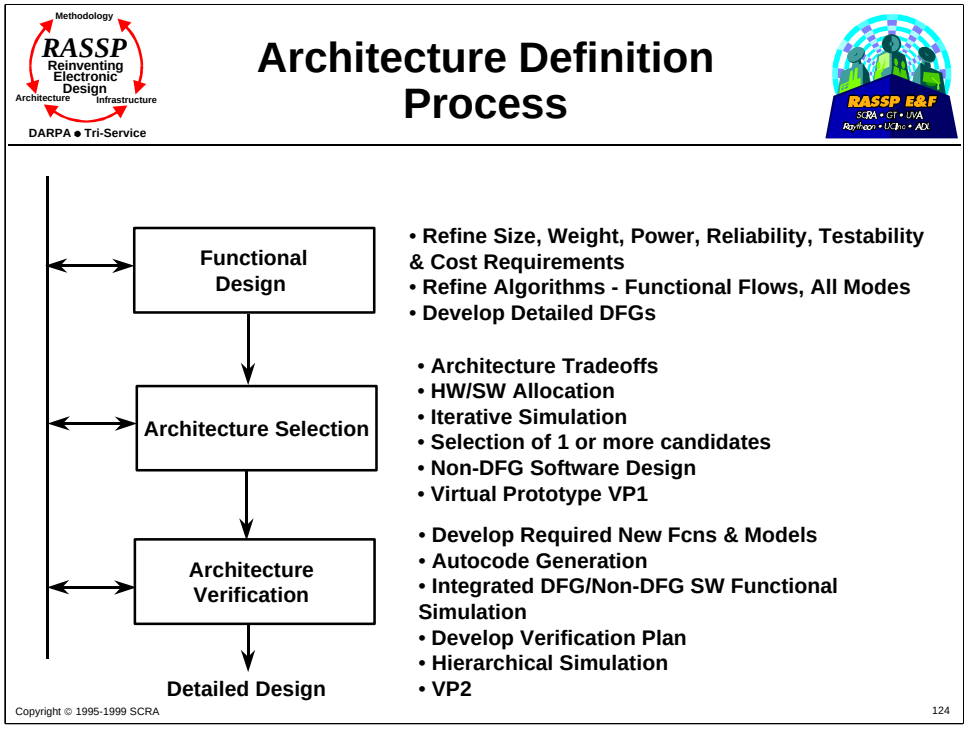
- **Considerable amount of progress has been made in developing techniques to translate PGM graphs into executable forms that minimize the amount of overhead associated with data flow paradigm**
- **These forms include**
  - **Sequential lists of primitive executions that execute a cycle of the translated graph**
  - **Time line programs**
  - **Parallel forms of list or time line programs**
- **Applications may be translated to**
  - **Completely static form**
  - **Hybrid forms of static representation of the segments of an application graph which may be dynamically scheduled by data flow scheduling of equivalent nodes replacing the segments in an equivalent graph**

See http://www.ait.nrl.navy.mil/pgmt/ for additional information on PGM.

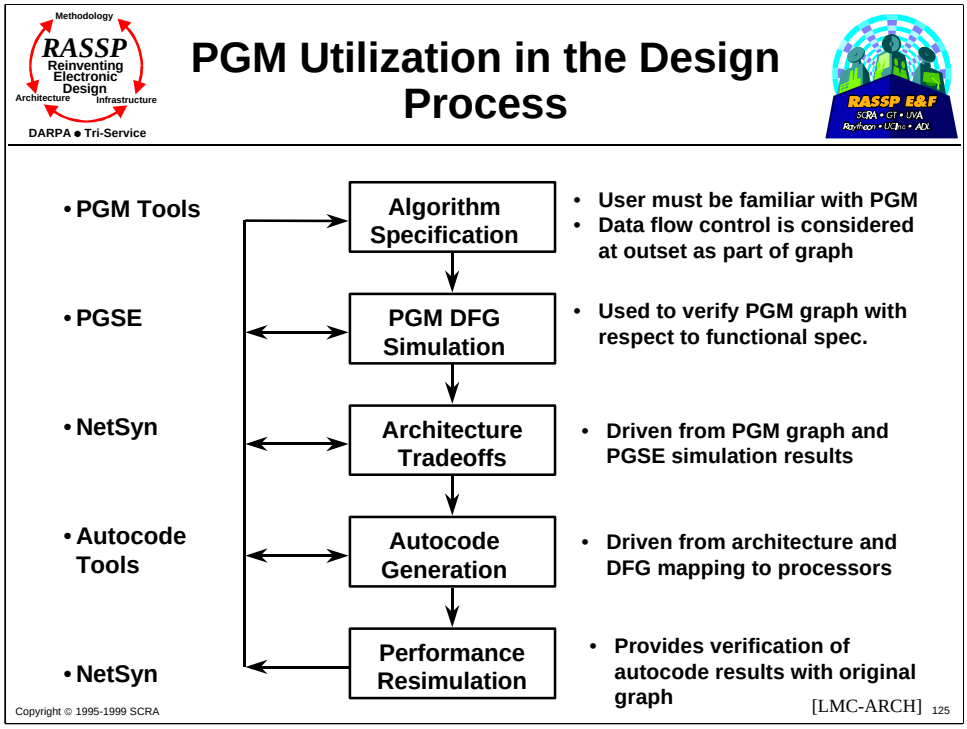
[MMC/LMC 94]

Module Outline

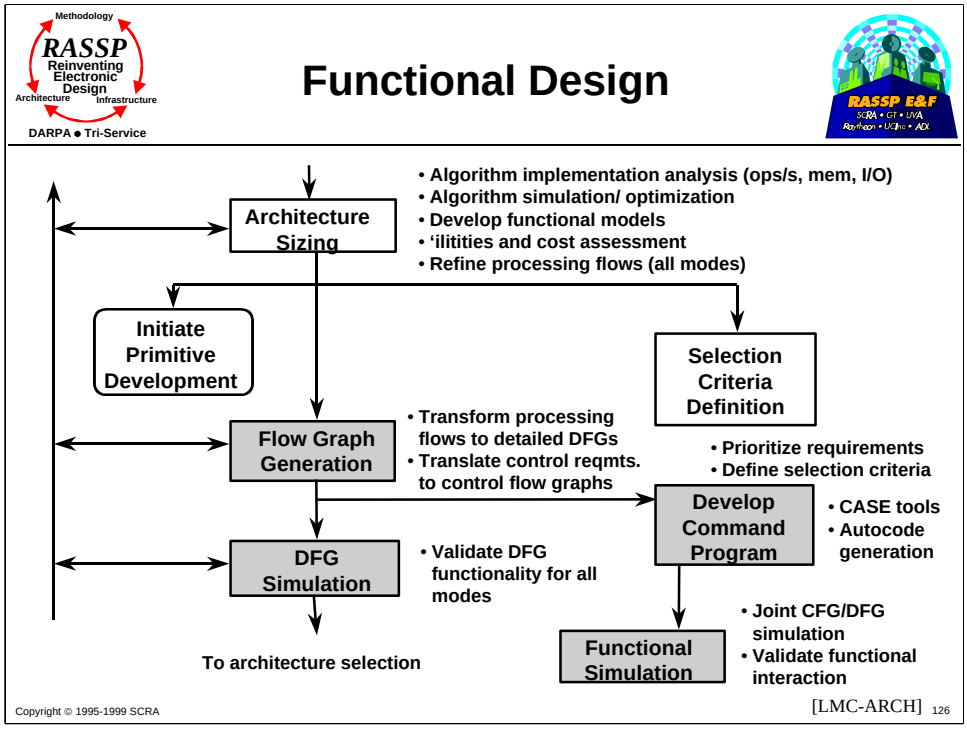
- Introduction
- Requirements Capture
- Fixed-Point Design - A RASSP Approach
- Simulation-Based Algorithm/Functional Design
- Network-Level DSP
- Link-Level DSP
- Signal Processing Simulators
- PGM/PGSE
- **RASSP Software Generation**
- Summary



The overall flow of the architecture definition process that begins with functional design and inputs into detailed design. PGM fits in well as a representation of input specification.



PGM is currently being used in the design process at the algorithm specification level. Data flow graphs are generated in PGM from Ada primitives. These are used to simulate the processing flows for the given application. The PGM primitives are simulated using the PGSE environment. After satisfactory simulation results are obtained, the PGM graphs are input to the NetSyn tool to begin architecture trade-offs.



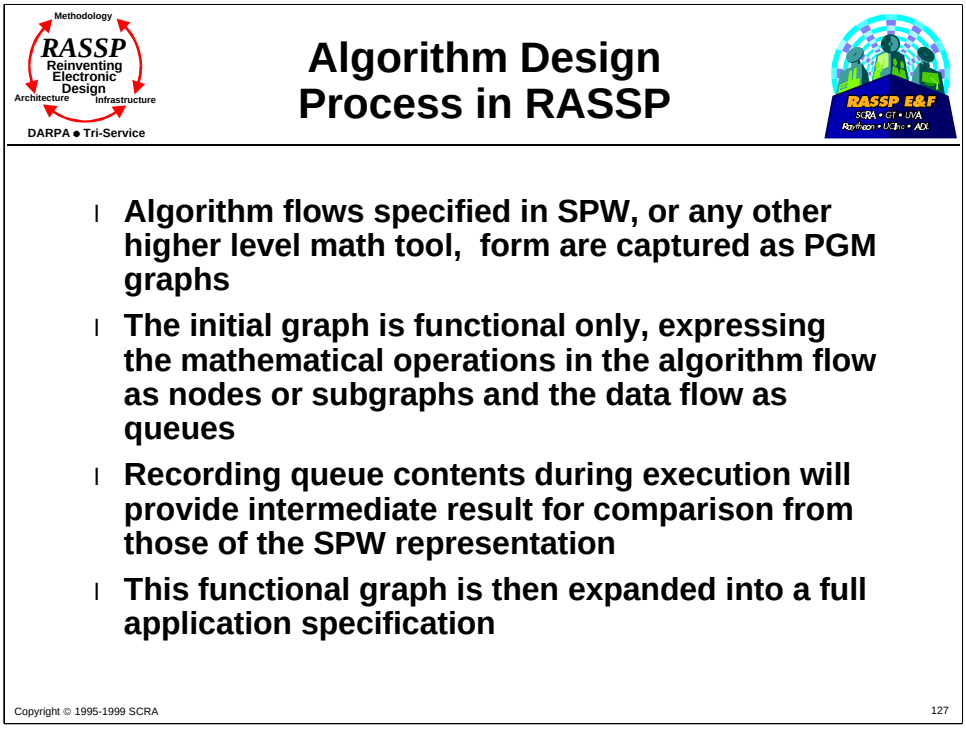
The functional design step provides a more detailed analysis of the processing requirements resulting in initial sizing estimates, detailed data and control flow graphs for all required processing modes to drive the HW/SW codesign, and the criteria for architecture selection. The control flow graphs provide the overall signal processor control, such as mode switching (referred to as the command program). Functional simulators support the execution of both the data and control flow graphs.

Architecture sizing helps to analyze the system requirements and processing flows for all the required modes of the system in terms of estimated operations per second, memory requirements, and I/O bandwidths.

Selection criteria definition helps prioritize the overall system requirements and the derived requirements and establishes a selection criteria. The selection criteria provides the necessary basis for subsequent architecture trade-off analysis. A trade-off matrix is used to formalize the selection criteria. It contains top-level requirements allocated to the signal processor.

Flow graph generation transforms the finalized algorithm processing flows into detailed DFGs as the first step in HW/SW codesign. The DFGs are based upon the Processing Graph Method (PGM) developed by the Navy. PGM is a specification for defining detailed DFGs for signal processing applications. The DFGs are made up of reusable library elements, which may represent either hardware or software. The DFGs are the basis for both the architecture synthesis, the detailed software generation, and potentially custom processor synthesis. Each DFG is simulated to provide data for comparison with the algorithmic flows developed during the systems process (executable spec). Control flow requirements are transformed into the control flow graphs (CFGs) required to manipulate the DFGs according to a defined set of rules. This DFG control is referred to as command processing. Conceptually, the command program manipulates objects. The objects are the DFGs and their data structures. The command program must be able to accept messages from outside the signal processor, interpret those messages, and generate the appropriate control information to stop graphs, start graphs, initiate I/O, set graph parameters, etc. The command program can be developed through standard software development CASE tools or through the tools that provide autocode generation capability.

Functional simulation verifies both the DFGs and the CFGs and their interrelationships.

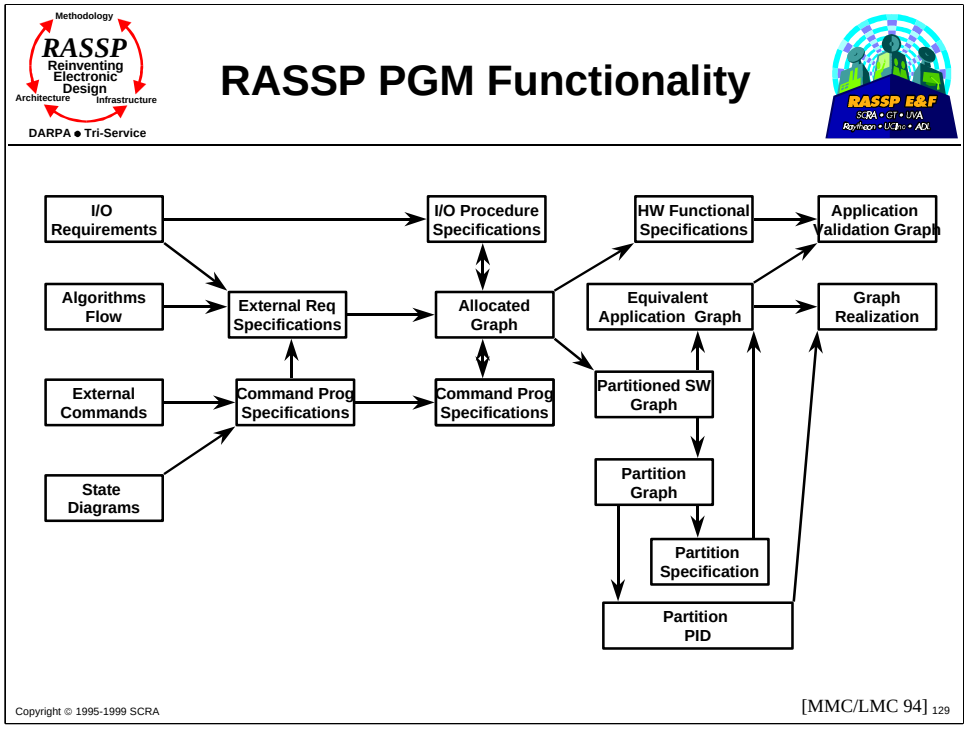


Algorithm design refers to functional design.

[MMC/LMC 94]

Application Development Process in RASSP

- Topology is expanded to represent
  - All possible data paths
  - Controls are specified for all operating modes
  - Sets or ranges of values for all variables are specified
- Graphical specification and execution tools are used to produce a full PGM specification of the applications
- Primitive support is target independent, allowing for the execution of domain primitives on general purpose hardware running the graph execution tool
- Provisional primitives may be added through library population for SPW functions for which no library support exist



Please refer to Lockheed Martin ATL documents for further discussion on how PGM was used to represent executable specifications.

Application Development Process in RASSP: System Definition

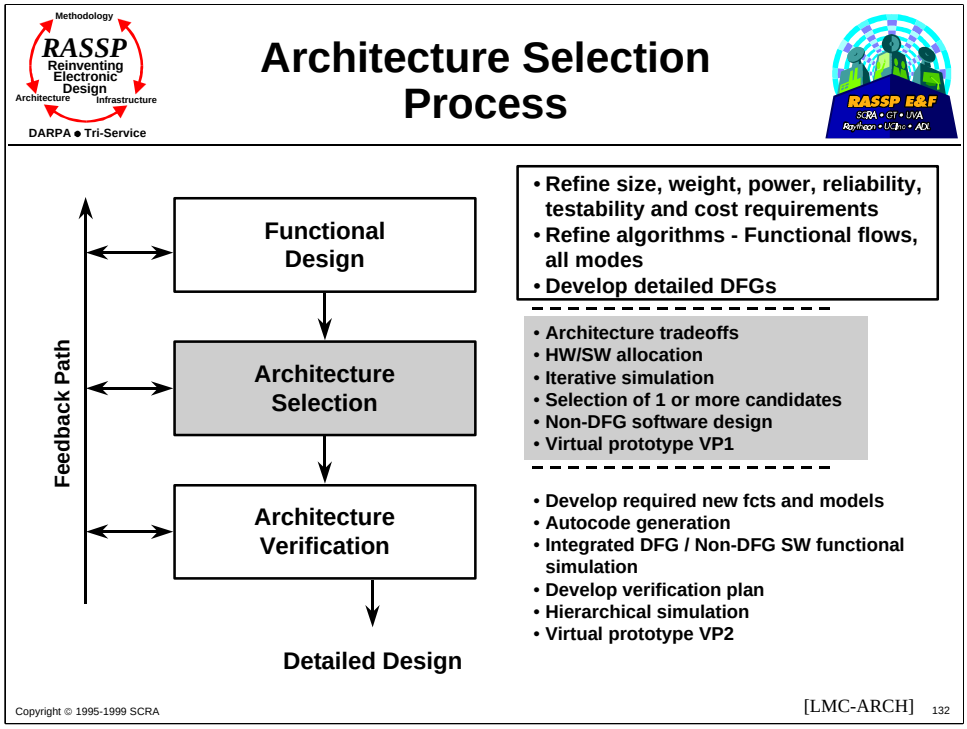
- **Algorithm flows**
  - **Are block diagrams depicting the high-level computations associated with an application, e.g. SPW, MATLAB**
- **I/O requirements**
  - **Identify the amount of throughput the application will have to handle, modes of data, and the source(s) of the data**
- **External commands**
  - **Are formal definitions of all commands allowable to the operator**
- **State transition diagrams**
  - **Depict the major modes of system operation, and the state changes between them. They, along with the external commands, are the basis for the command program specifications**

[MMC/LMC 94]

Application Development Process in RASSP: Requirement Refinement

- **Executable Requirements Specification**
  - **A PGM graph is composed of domain primitives, domain subgraphs, provisional primitives or subgraphs or any entity which can specify the workings of an algorithm**
  - **Command Program Specifications**
  - **Derived from the state transition diagrams and the external definitions defined in the previous phase**

[MMC/LMC 94]



The architecture definition process transforms processing requirements into a candidate architecture of hardware and software elements.

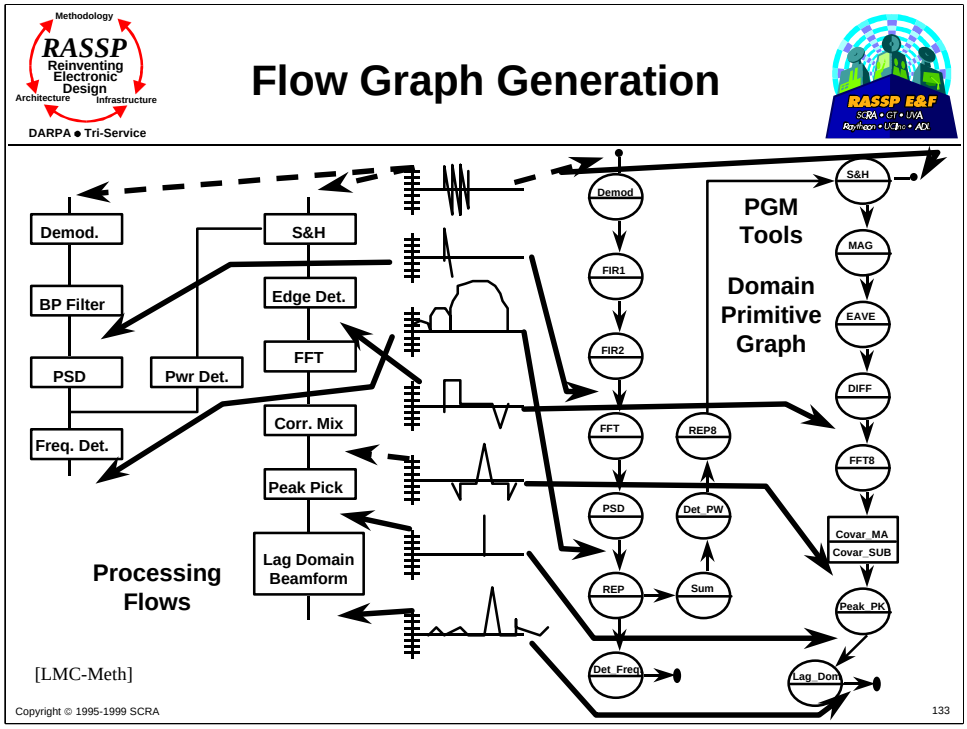
The architecture definition process is a new HW/SW codesign process in the RASSP methodology for high-level virtual prototyping and simulation. The primary concern in the architectural definition process is to select and verify an architecture for the signal processor that satisfies the requirements passed down from the systems definition process.

The overall task is to:

- Define and evaluate various architectures
- Select one or more for detailed evaluation that appear to meet the requirements
- Validate the chosen architecture(s) for both function and performance before detailed design

Concurrently, each selected architecture is evaluated with respect to size, power, weight, cost, schedule, testability, reliability etc.

The process is library based and DFG-driven. The DFGs are created from the processing flows passed down from the systems definition process. Reuse of both architecture elements and software primitives significantly shortens the design cycle. VHDL performance model simulations are used to verify system requirements are met. Software performance is also modeled for its impact on the total processing time.



The transformation of the finalized algorithm processing flows into the detailed DFGs is the first step in the HW/SW codesign process. These DFGs are based upon the Processing Graph Method (PGM) developed by the Navy. PGM is a specification for defining detailed DFGs for signal processing applications. The DFGs are made up of reusable library elements, which may represent either HW or SW.

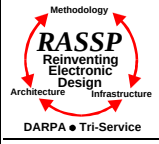
The DFGs are the basis for architecture synthesis, detailed software generation, and potentially custom processor synthesis.

The left side of this figure represents the processing flows as passed down from the systems definition stage. The right side represents the detailed DFG constructed from reuse library elements.

If suitable library components do not exist, then they need to be developed and added to the library.

Application Development Process in RASSP: Architecture Selection/Verification

- **Domain Primitive Graph**
  - **Is the executable requirement specification refined to contain only domain level primitives and subgraphs**
  - **Becomes the functional baseline graph for the application once it has been validated with the test vectors developed for the executable refinement specification**
  - **Allocated Graph**
  - **Produced by assigning each domain primitive node or subgraph of the domain primitive graph onto hardware or software components**



Application Development Process in RASSP: Architecture Selection/Verification (Cont.)

- **Partitioned Graph**
  - **Is the state of the domain primitive graph after the partitioning process**
  - **Is not a separate graph, but merely the domain primitive graph with partitioning information**
- **Partitioned Software Graph**
  - **Is the subset of the partitioned graph which will be used by the software development team**
  - **May be partitioned separately from the full partitioned graph so that software partition performance may be optimized**
  - **Criteria to be used in determining the optimal scheme are:**
    - **The best processor class for each partition in the graph**
    - **The selection of run-time system to be used based on the characteristics of the application, i.e. static, hybrid, etc**

[MMC/LMC 94]

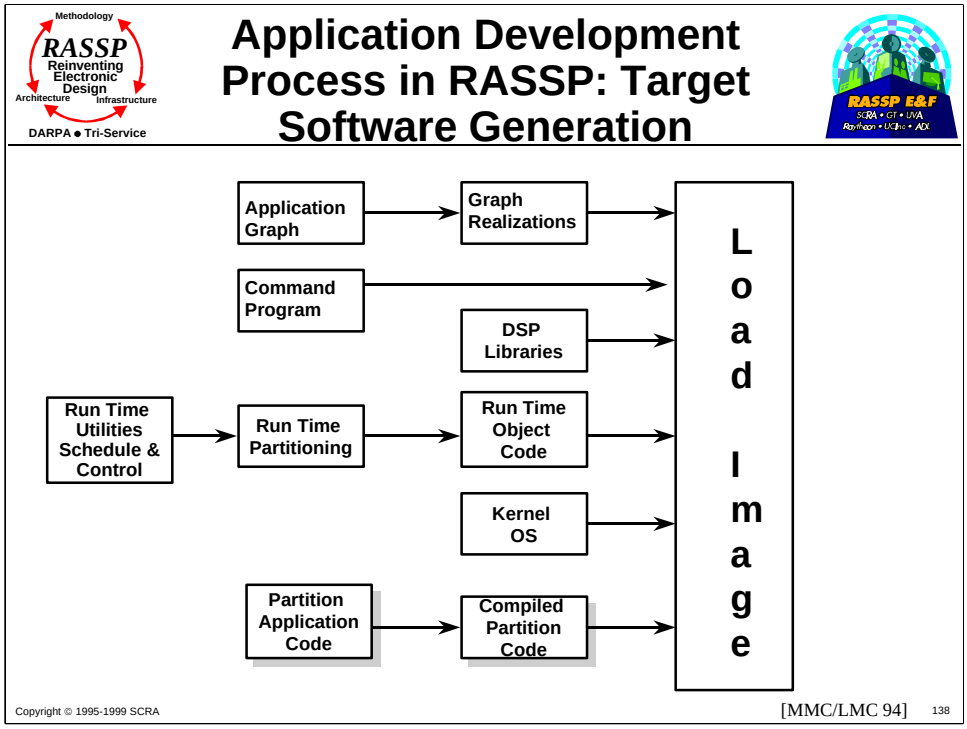
Application Development Process in RASSP: Architecture Selection/Verification (Cont.)

- **Partition Graph**
  - **Is a stand-alone SPGN graph which represents either a hardware or a software partition**
- **Corepresentation Validation Graph**
  - **Is a PGM graph in which all partitions identified in the partitioned graph have been replaced with equivalent nodes**
  - **This graph is used to simulate the functionality and performance of the entire application graph on selected candidate architecture**
  - **Partition Specification**
  - **For each partition graph, a partition specification is produced providing an interface to the partition PID which is produced to represent the partition graph**

Application Development Process in RASSP: Target Software Generation

- **Equivalent Application Graph**
  - **Is produced from the partitioned software graph by replacing each software partition with an equivalent node produced by Multi-processor Primitive Interface Description (MPID) generator**
  - **Graph Realization**
  - **Is the compiled version of the equivalent application graph**
- **Partition PID**
  - **A partition PID is the source code that represents a partition graph, written in the computational element's native language**
- **Load Image**
  - **Is the end product of software development process**

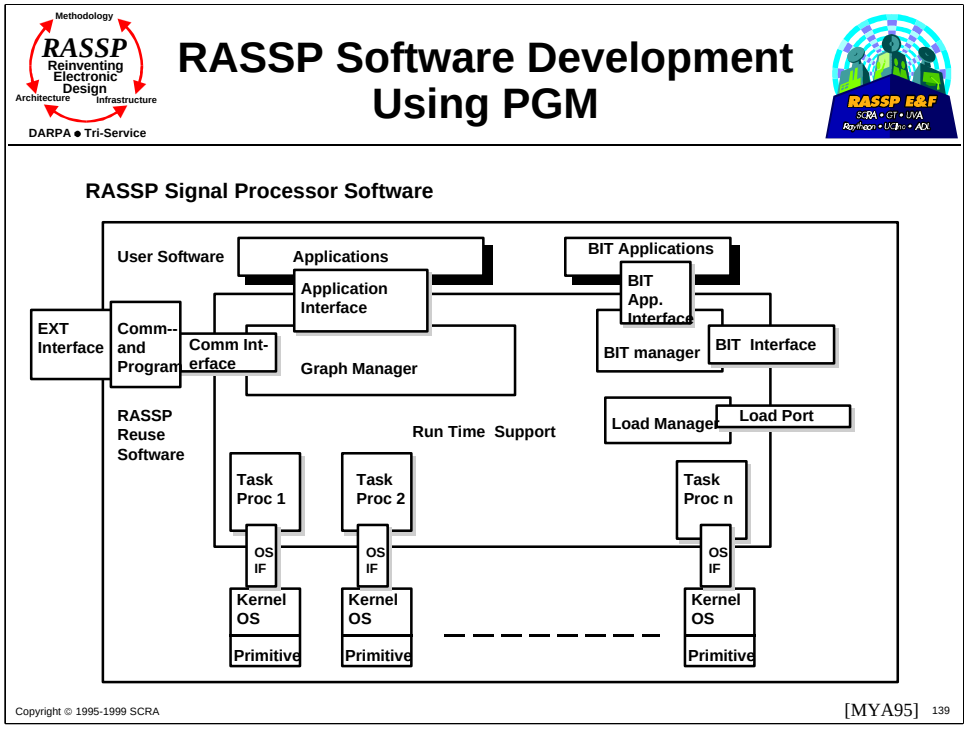
[MMC/LMC 94]



Description:

The figure shows all the elements that are needed to obtain a load image of the application.

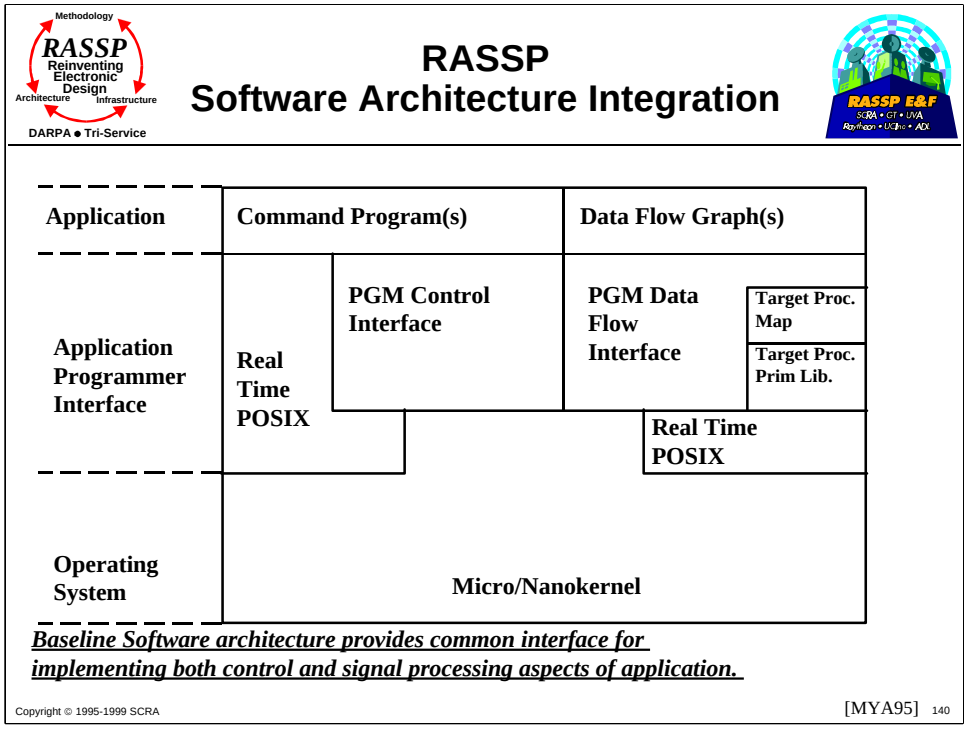
Top level application graph SPGN specification is compiled using the graph realization. The DSP libraries provide object code for specific implementation of some of the primitives . All run-time utilities, schedule and control routines are also partitioned and then presented to the load image box to be integrated with the application code. Kernel OS is also presented to the load image, the run-time functions use the OS calls. The partitioned application code is also compiled and then presented to the load image.



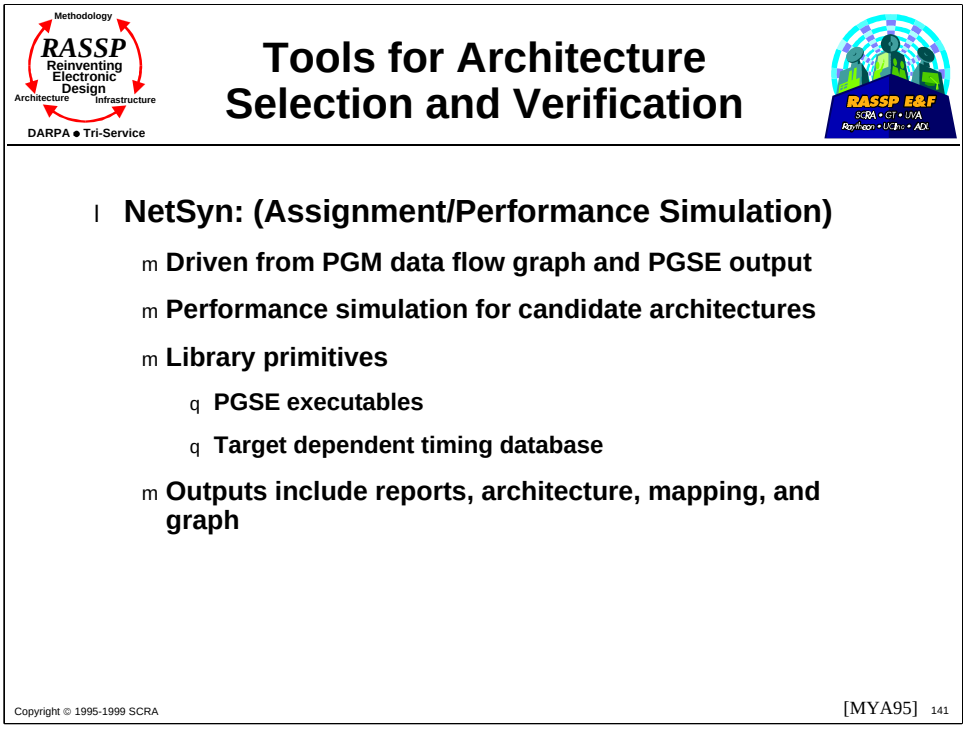
Description:

The figure shows the software architecture of RASSP signal processing application running on a multi-processor platform.

The external interface can choose among a number of command programs. Based on the command program, application graphs are loaded on the multiprocessor platform based on the partitioning information that is available for each application. Based on the command program the graphs and the I/O procedure are started through the run time support.

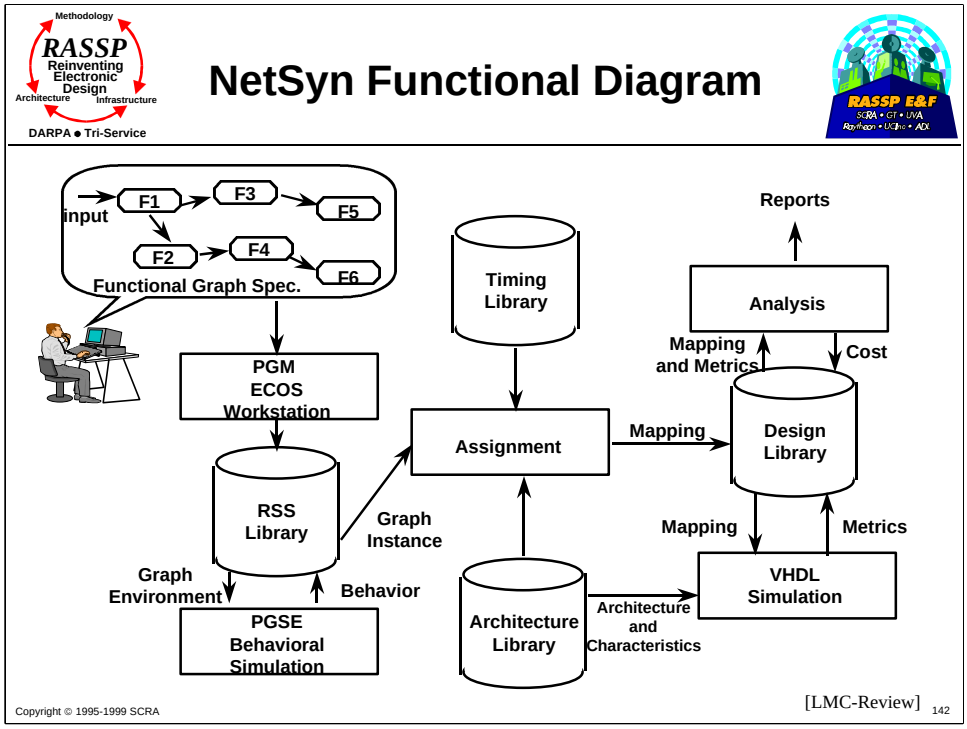


This architecture is similar to the virtual machine.

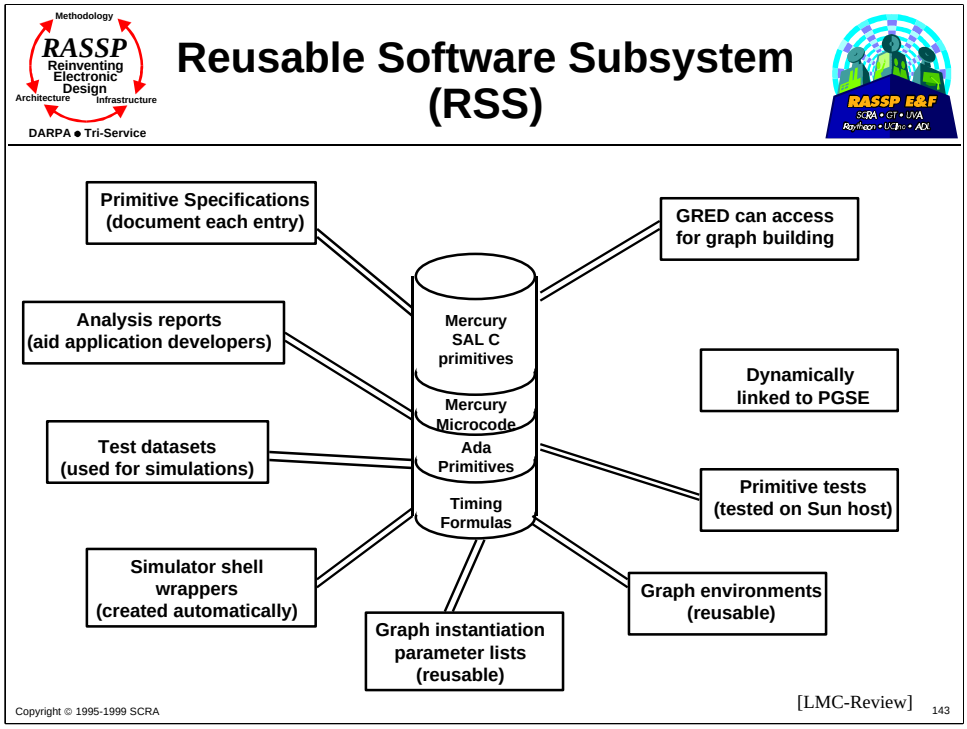


[LMC-ARCH]

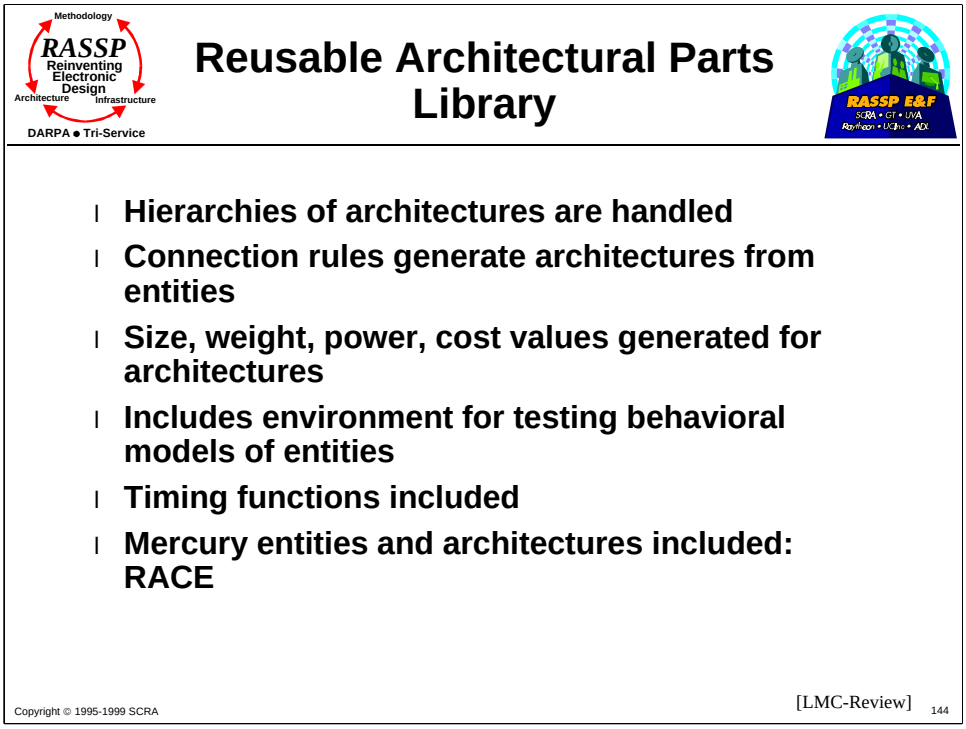
NetSyn allows performance trade-offs to be done in a more automated/user friendly fashion. The input to this tool will be the PGM data flow graph and PGSE output. Rapid performance trade-offs are made within the environment by choosing candidate architectures and simulating them using performance models from Honeywell. Outputs include reports, improved architectural candidates, SW mappings, and improved flow graphs.



The NetSyn tool contains three reusable parts libraries to aid in the selection and verification of an architecture. They consist of 1) a Reusable Software System (RSS) which contains functional graph primitives to help in the building of applications, 2) a reusable architectural parts library which contains architectural classes, components, and configurations, capable of being simulated at the performance level in VHDL and 3) a timing library which contains timing information for the execution of the specific primitives on various processors.

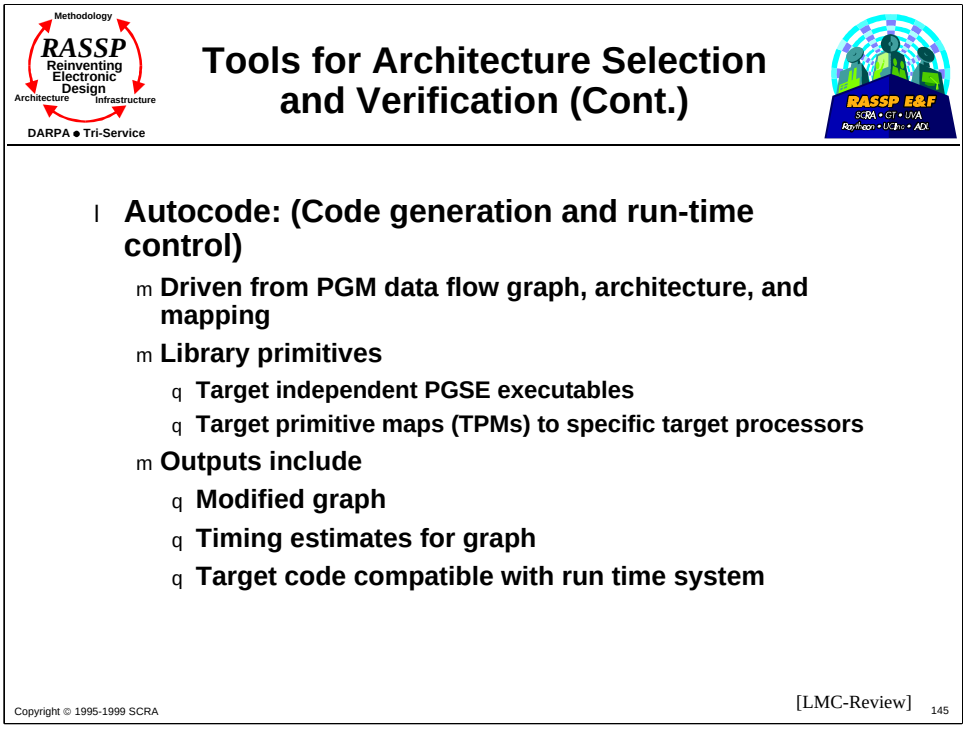


The Reusable Software Subsystem (RSS) captures functional graph primitives in the form of C, Ada, Microcode etc.. It also captures test datasets, analysis reports to aid application developers, reusable graph instantiation parameter lists, reusable graph environments, and primitive tests. The graphical editor for generating PGM graphs (GRED) can access the primitives for building new graphs.



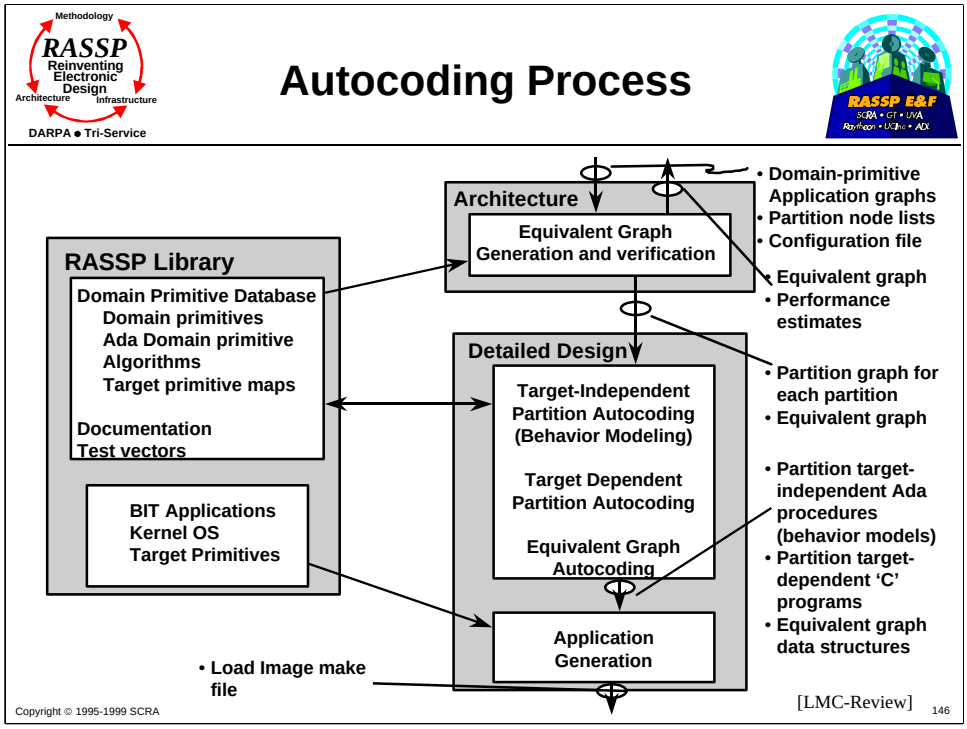
[LMC-ARCH]

The reusable architectures are hierarchically composed. Connection rules are used to rapidly generate architectures from entities. The performance models use size, weight, power, and cost values so the tool can quickly generate estimates for the entire system. The environment includes capabilities for testing the behavioral models of entities. Timing is included in the library for each of the parts. Currently, the complete RACE architecture from Mercury is part of this library.



[LMC-ARCH]

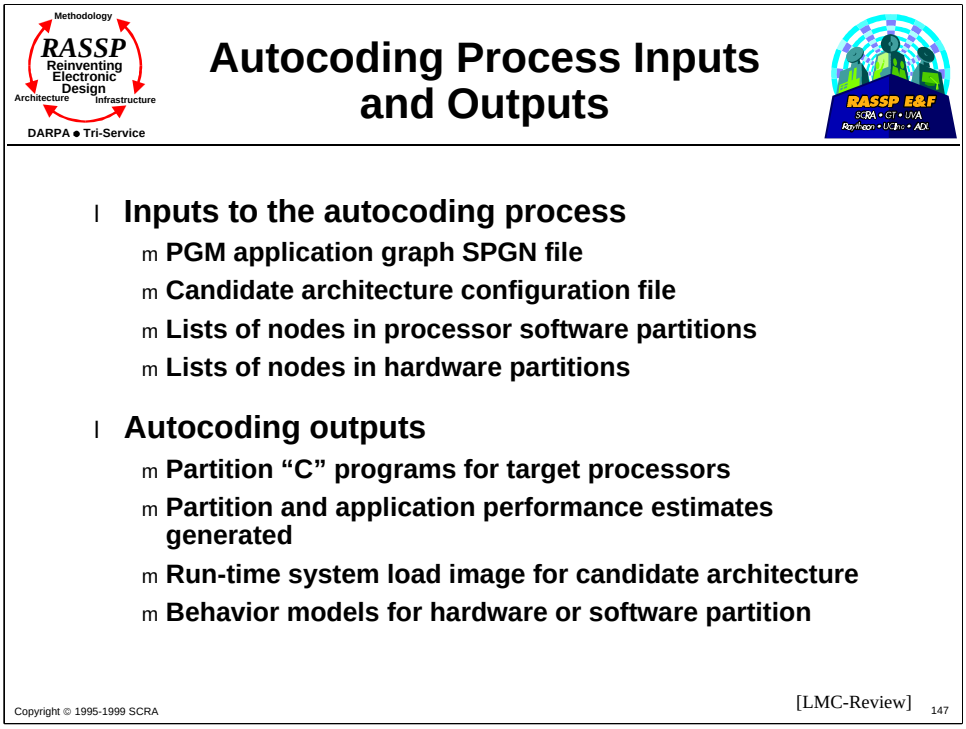
Autocode is a tool that takes as input, PGM data flow graphs, and generates a modified graph with updated timing estimates for the code which has been mapped to a target processor. The target code is compatible with the run-time system. The next slide shows more detail of the autocode generation process.



The autocoding process is shown above. The main elements of the process include:

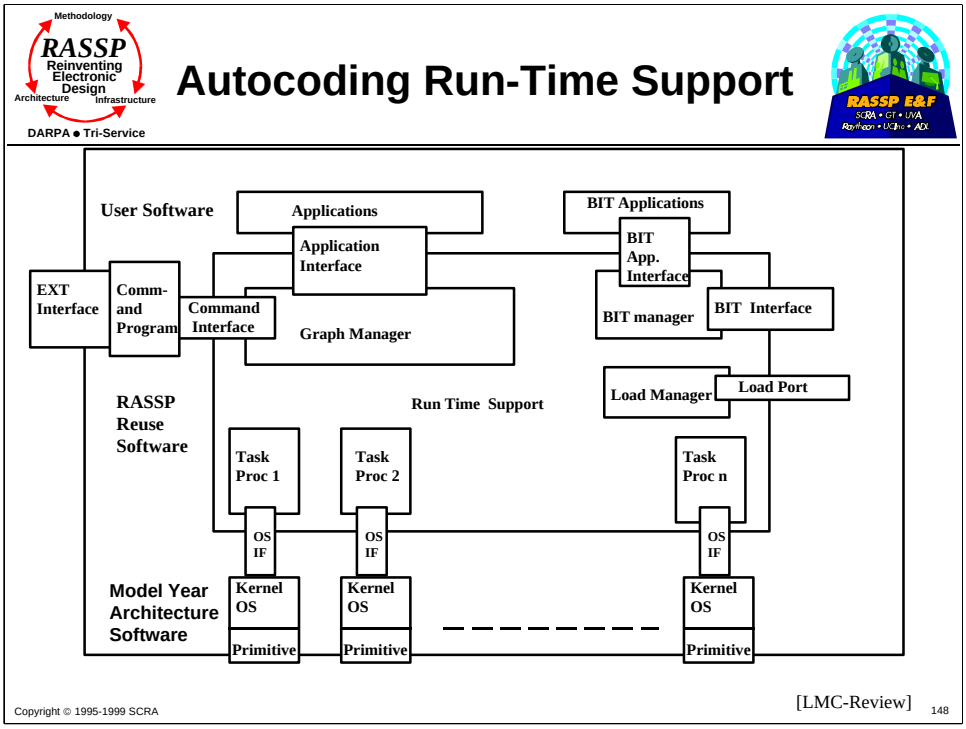
- Equivalent graph generation
- Partition target-independent autocoding
- Equivalent graph autocoding
- Partition target-dependent autocoding
- Load image specification

The inputs to the equivalent graph generation process are domain-primitive graphs, configuration files, and partition lists. Equivalent graph generation generates standalone PGM graphs for each partition. Partition autocoding generates Ada procedures implementing each partition (behavior model) and 'C' programs implementing each software partition using target math libraries. Equivalent graph autocoding creates run-time data structures implementing the equivalent graphs. Load image specification generates "make" files specifying complete run-time system.

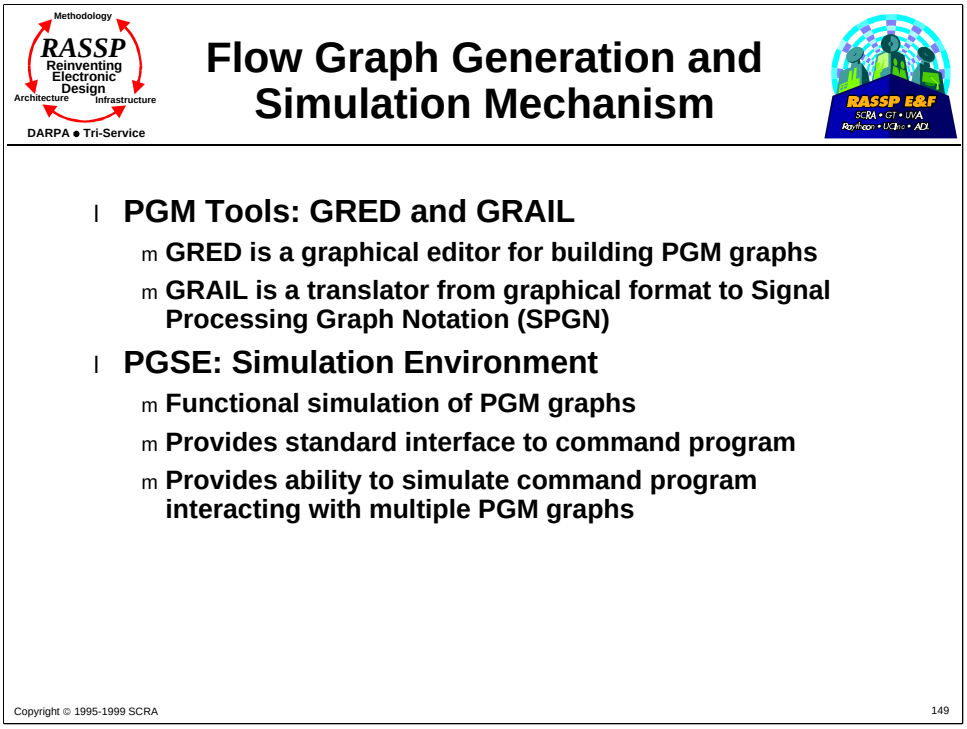


[LMC-Review]

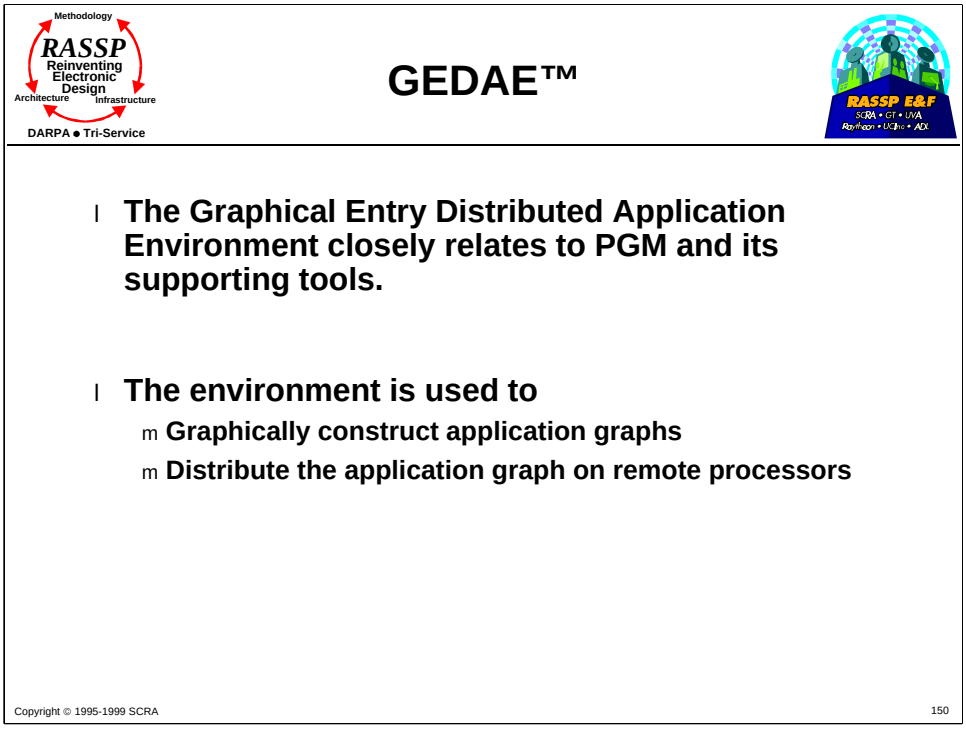
This slide lists the inputs and outputs of the autocoding process.



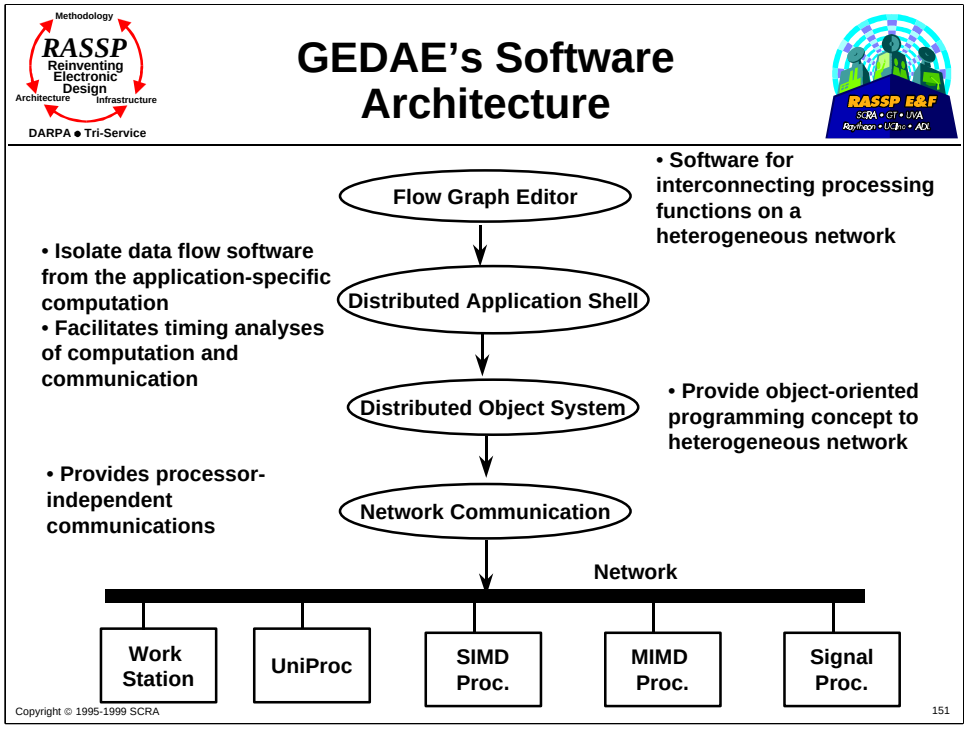
Run-time support is partitioned into user, RASSP user, and Model Year Architecture parts. There are driver-level interfaces between the reuse and model year partitions. Application interfaces are isolated from the target OS. Ports to the external world include the load port, BIT interface, and the command interface. The load manager, graph manager, and BIT manager are run-time managers that execute run-time service routines. Applications are instanced as equivalent node tasks and multiple instances, priorities, and preemption are possible.



The main tool used by the ATL branch of Lockheed-Martin is the PGM tool developed by the Naval Research Labs. PGM is used to model the data flow of the system and contains hundreds of primitives written in Ada to develop algorithm designs. GRED and GRAIL are graphical tools to aid in PGM development while PGSE is the simulation environment used to perform the functional simulation on PGM graphs.



Please refer to http://www.gedae.com for additional information. GEDAE is a tool developed by Lockheed Martin ATL on the RASSP program and the next few slides describe the algorithm design and mapping process using GEDAE.



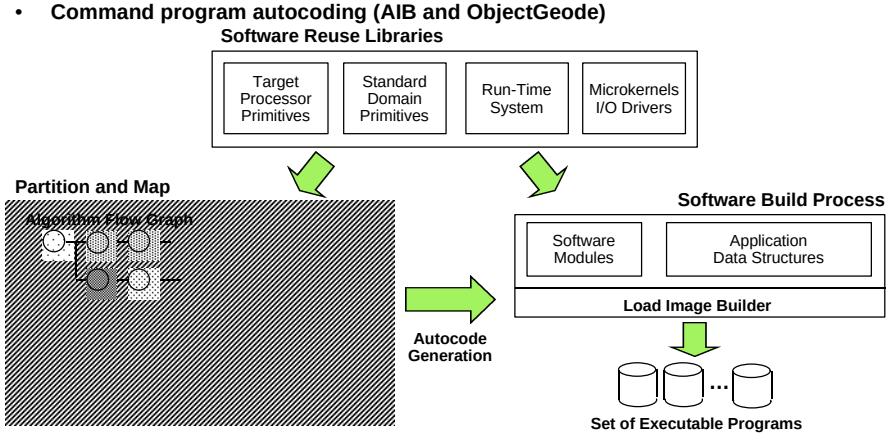
Please refer to http://www.gedae.com for additional information.

A Case Study of RASSP Software Development Using GEDAE

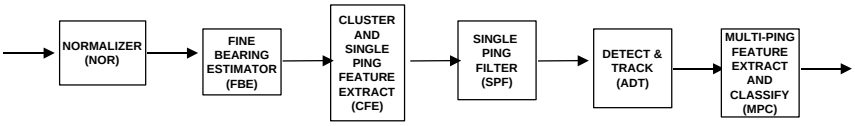
- **Reduce development/lifecycle cost of DSPs**
- **Minimize hardware dependence**
  - **Enable rapid porting to new hardware**
- **Maximize software reuse**
  - **Minimize rework between algorithm stage and system implementation**
- **Enable rapid HW/SW tradeoffs of candidate system architectures prior to implementation**
  - **Enable rapid repartitioning and assignment of functionality between microprocessors**
  - **Minimize rework for system upgrades**

RASSP Software Generation

- **RASSP graph-based autocode generation (GEDAE) capability significantly improves software productivity**
- **Supporting Run-Time System provides all graph management, interprocessor communication, and external interface**
- **Command program autocoding (AIB and ObjectGeode)**

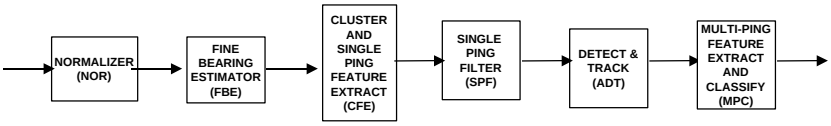


ETC Algorithm Description



- **Front-end Processing (Acoustic signal processing)**
  - **Normalizer (Norm)- compensation for underlying noise background, dynamic intensity variations such as may occur in reverbation-limited shallow water.**
    - **The Coded Pulse (CP) normalizer is a single dimension normalizer, range only**
    - **The Coherent Waveform (CW) normalizer is a two dimensional normalizer, range and Doppler**
  - **Fine Bearing Estimator (FBE)- high resolution bearing information**
    - **Interpolation between beams with Doppler compensation**
  - **Clustering and single ping Feature Extraction (CFE)**
    - **The data is first thresholded and then the single celled clusters are merged into multi-celled clusters according to their proximities to one another. Single ping features are extracted from each cluster.**

ETC Algorithm Description



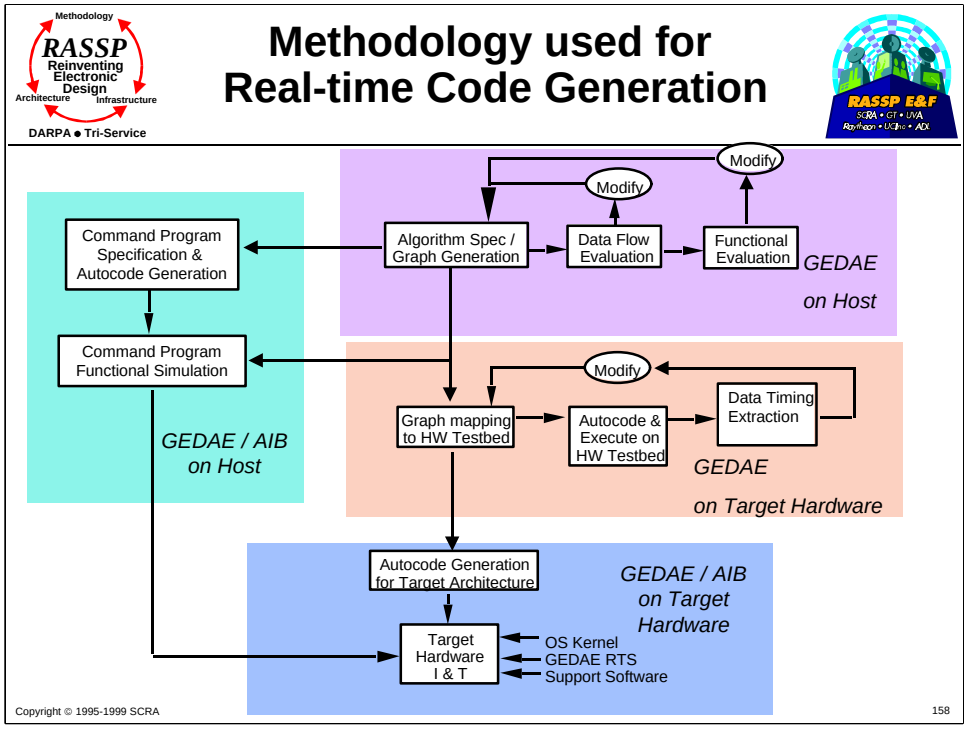
- **Back-end Processing (detection and classification)**
  - **Single Ping Cluster Filtering (SPF)- weed out some of the echo returns from clutter**
  - **Automatic Detection and Tracking (ADT)- automates the detection of potential targets over multiple pings.**
  - **Multi-ping Classifier (MPC)- classifies each track into three possible states, sub, non-sub or pending.**

Legacy Software - Converting Lab Code into a Real-time Embedded System

- Laboratory C/Matlab code delivered as executable specification
  - Messaging backplane for data and parameter distribution
  - Large use of gobal variables
  - Non-segmented processing on a per ping basis
  - Combination of data and control flow
  - ~50,000 lines of code
- Migration
  - Analyzed c-code hierarchy (in-house tools)
  - Extracted core processing code
  - Encapsulated code into GEDAE at highest level
  - Decomposed only high load DSP functions for distributing
  - Parallalize and segmentize signal processing code
  - Graph results were required to match within 10E-5 of output of NUWC simulation at each stage of output, this required more custom primitives to be generated
- Matlab called directly from GEDAE
  - Supported migration while on development host

What This Team Considered As Critical Contributors to Success

- **Modern graphical design and development environment**
  - **Easy capture of graph and fexibility to modify**
  - **Easy finding of primitives for reuse**
  - **Easy addition of new primitives**
    - **Tools being used by new users**
  - **Good error checking as graph is built**
- **Robust, visually rich test and integration environment**
  - **Unified environment enabling seamless design flow from functional/data flow simulation to autocode and multiprocessor distribution**
  - **Good error comments from tools and help in debugging process**
  - **Good vizualization aids to eliminate problems of "black box" effects of using code not completing generated by team on this project**

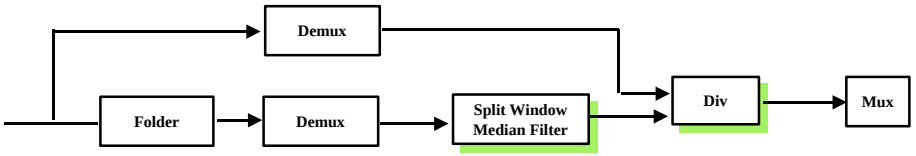


Graph Development Process

Graph Generation

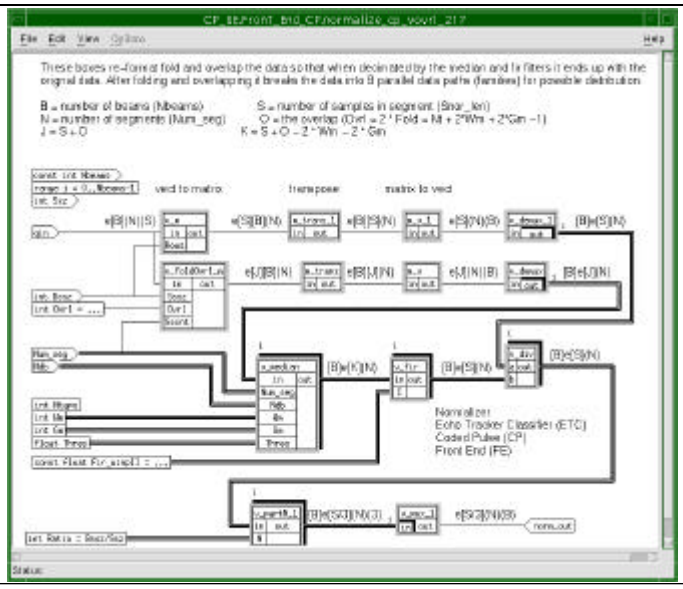
- **Determine Data Flow**
  - Scalar stream, vectors, matrices, variable vectors/matrices, structures
  - Instantiation
  - Queue sizing
- **Determine Functional Flow**
  - Custom vs. built-in primitives
    - Tradeoffs include - entry time, readability, maintainability, debug and test, runtime performance, portability, distribution
  - Determine baseline hierarchy
- **Write Custom Primitives**
  - Written in portable math library (e-library)
  - Can make direct call to custom c-function
- **Enter Graph and Parameters**

Norm CP Data Flow Example



- Use non-deterministic box to support variable threshold, and variable consume and produce for variable segmentation
- Use family construct to parrellaize numerically intensive processing
- Use V-arrays to support parameter dependent variations in data flow

Norm CP GEDAE Graph



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

Addition of new custom processing primitives

- **Data transfer separated from application module design**
  - — programmer only concerned with application specific code
- **Data flow specification separated from application module**
  - — programmer provides simple specification wrapper to encapsulate module

Module Code

Data Processing Function

split (in, re, im) {

derive output arrays ("re" and "im") according to some algorithms applied to input array ("in").

}

Standard Interface

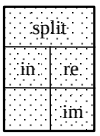
Data Flow Requirements

applysplit ( ) {

1. Get inputs and outputs
2. Call split data processing function
3. Note data produced and consumed

}

GEDAE Function Box



GEDAE™ CODE Generation

Library Function
embeddable/stream/add

```
Name: add
Type: static
Comment: "Addition
out = a + b"
Input: {
  stream float a;
  stream float b;
}
Output: {
  inplace stream float out=a;
}
Include: {
#include <e_vadd.h>
}
Apply: {
  e_vadd(a,1,b,1,a,1,size(a));  /* vector add */
}
```

Automatic Code Generation →

C source of add.c

```
#include <stddef.h>
#include <staticFunction.h>
typedef struct {
  float *a;
  int N_a;
  float *b;
  int N_b;
} StateRec, *State;

static OffsetTable OT[] = {
  {VALUE_OFFSET, "a"},  {SIZE_OFFSET , "a"},
  {VALUE_OFFSET, "b"},  {SIZE_OFFSET , "b"},
  {0,0}
};
static int Offsets[] = {
  offsetof(StateRec,a), offsetof(StateRec,N_a),
  offsetof(StateRec,b), offsetof(StateRec,N_b),
  -1
};
static int Offset(char *name, OffsetType type) {
  return internal_offset(name,type,OT,Offsets);
}
#include <e_vadd.h>
static void Apply(void *vstate) {
  State state = vstate;
  float *b = (state->b);
  float *a = (state->a);
  int _size_a = state->N_a;
  e_vadd(a,1,b,1,a,1,_size_a);
}
void Init__embeddable__stream__add(void) {
  OCreateStaticBoxClass("embeddable/stream/add",sizeof(StateRec),
                        "offset", Offset,
                        "apply",  Apply,
                        0);
}
```

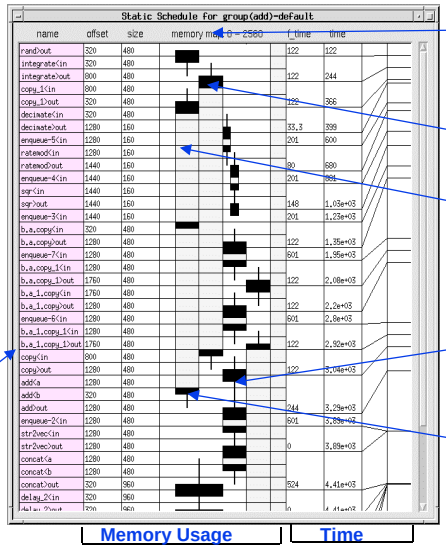
GEDAE™ library functions are simple to add and generated source code is not difficult to read

Test and Integration Process

- **Test Data Flow on Host**
  - **Primitive connection ensures data flow compatibility prior to test**
  - **Box table/data table, show Q's and boxes graphically facilitate debug of data flow**
  - **Optimization of memory**
- **Test Functional Flow on Host**
  - **Scope and/or files facilitates debug of functional flow**
  - **Standard debuggers used to debug custom c-code**
- **Collect Timelines Host**
  - **Identify processing requiring further optimization**
- **Embed on target hardware verify performance**
  - **Optimize partitioning and mapping for load balancing**
  - **Optimize transfer methods**
  - **Identify processing requiring further optimization**

Data Flow and Memory Usage Visualization

- **Static Schedule Table displays detailed information about statically scheduled processes**
  - execution sequence
  - memory usage vs. time
  - process execution time



Time Line Visualization

- **GEDAE provides visualization from both hardware and software perspectives**
- **Trace Table displays detailed timelines and processor loading for each function and data transfer**

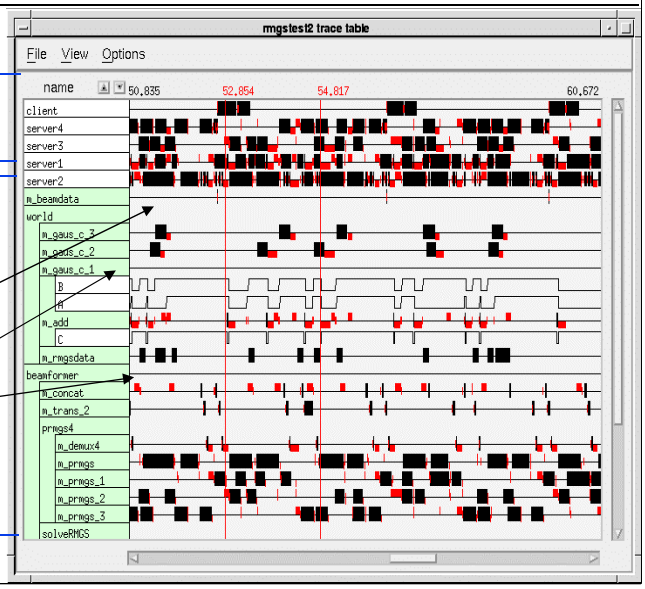
Hardware profile

Compute time

Data flow profile

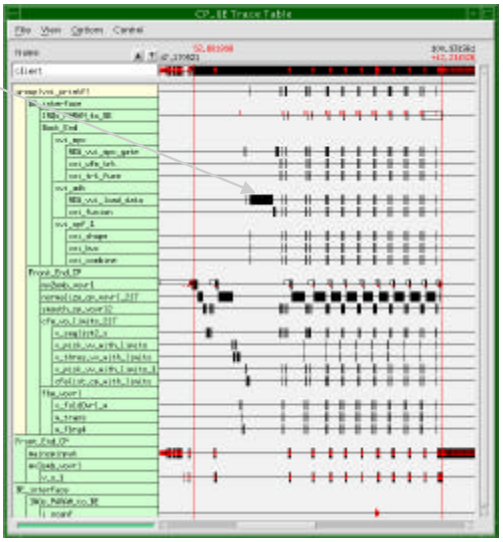
I/O time

Software profile



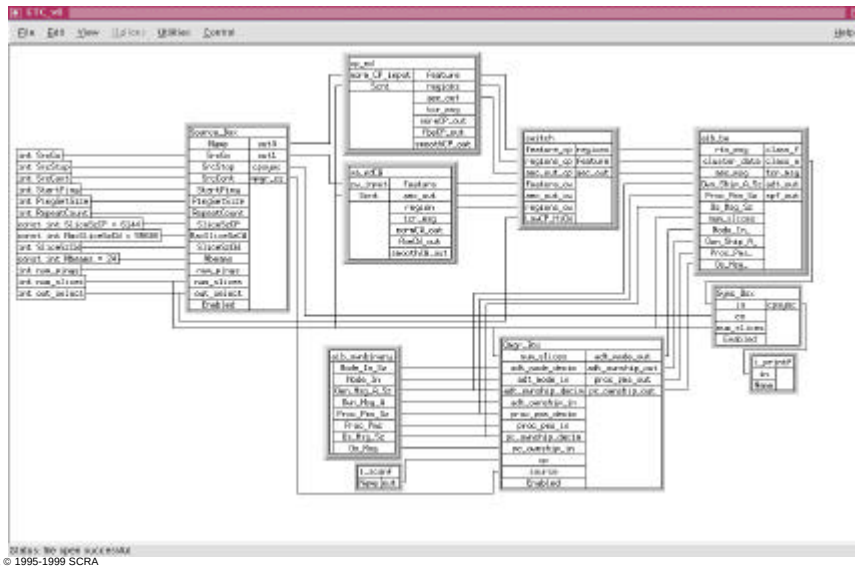
ETC4ALFS Timeline Example

- **Timeline of graph running on host**
  - **First run identifies a problem**
    - **ADT initialization routines need to be done on reset method at graph initialization time**



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

Top Level Graph

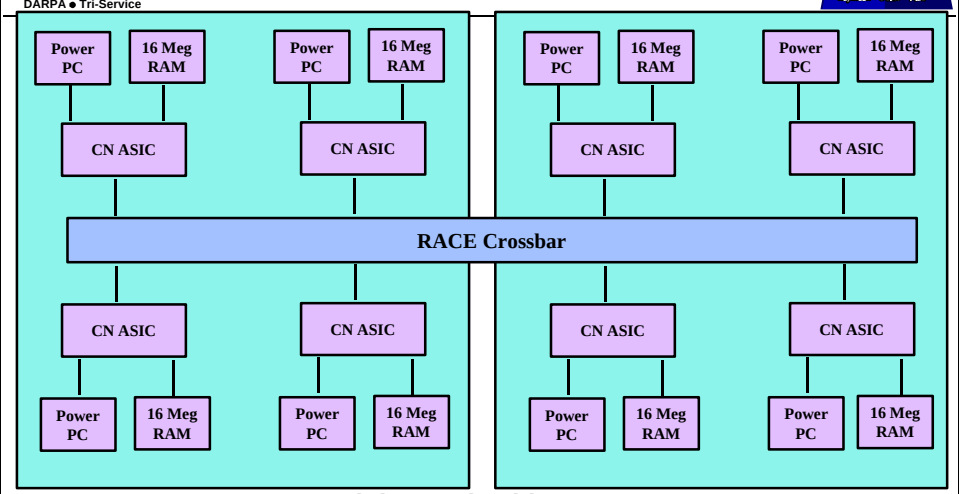


ETC4ALFS Top Level Graph

The graph is composed of

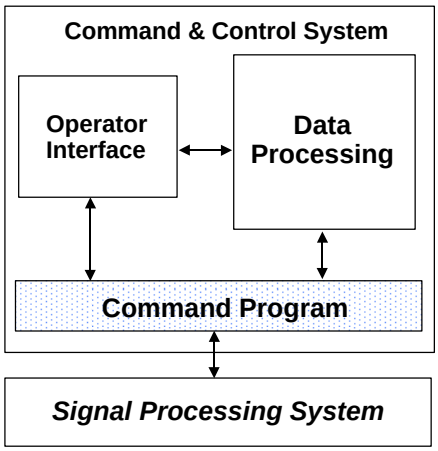
- Source_Box which simulates real-time input data
- The Front end CP (cp_nd) composed of
  - Norm,
  - FBE
  - CFE.
- The Front end CW (va_ndCW) composed
  - Norm
  - FBE
  - CFE.
- A switch to control flow of data to backend graph
- The Back end composed of
  - SPF,
  - ADT
  - MPC.
- QueueManager box to sync up beam data to ping time data and other data from Command Program
- Sync_Box to tell the command program when the graph is complete with each ping of data

Hardware Architecture Description



- Two Mercury modules each with 4 Power PC processors
- Command Program on Sun workstation

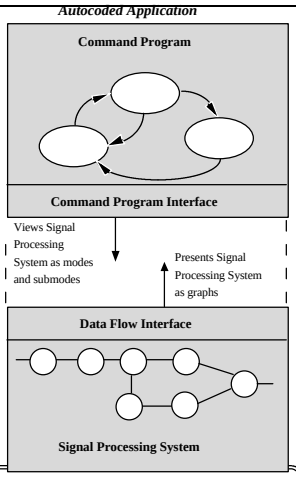
Command Program



- **Fundamental Command Program Requirement**
  - **Translate high level (user) input (e.g., stop track model, start weather mode) into lower level signal processor control commands (e.g., create queue, write graph variable)**
  - **Forward signal processor results**

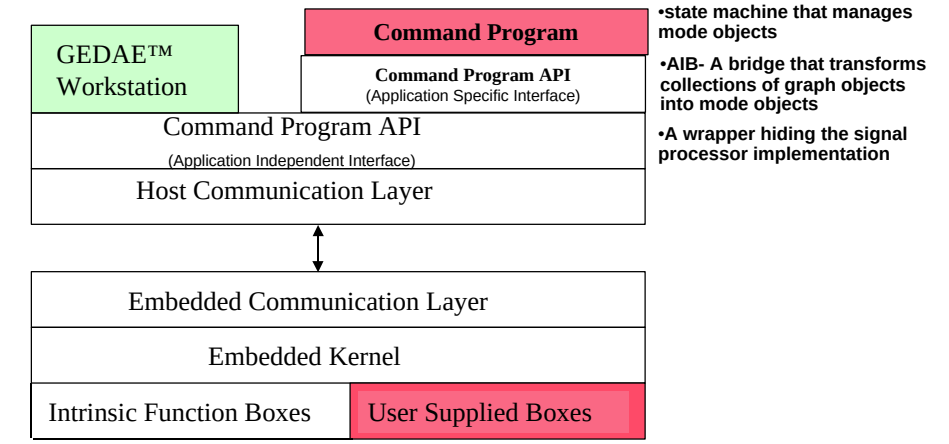
AIB - Application Interface Builder

- **Builds Application Programmer Interface (API)**
  - **Each mode is an object**
  - **Methods for mode entry/exit**
  - **Methods for submode entry/exit**
  - **Methods for execution state transition**
  - **Methods for buffering operator/DSP data**
- **Input**
  - **Enhanced top level graph declarations**
  - **90% previously generated by DSP autocode process**
- **Output**
  - **C source**
- **Future**
  - **Integrate with GEDAE GUI**
  - **Tighter integration with ObjectGeode**



Used successfully to support ETC for ALFS integration - 300 lines in yielded 3000 lines out

Layer and Conquer



- **state machine that manages mode objects**
- **AIB- A bridge that transforms collections of graph objects into mode objects**
- **A wrapper hiding the signal processor implementation**

ETC GEDAE Conversion Summary

- **Training - 1 week**
- **CP frontend captured, tested, and optimized on Host - 3 weeks**
  - **CP Front End is well partitioned into primitives**
- **CW frontend captured, tested, and and optimized on Host - 5 week**
  - **CW Front End is well partitioned into primitives**
- **Back End captured, tested, and optimized on Host - 2 weeks**
  - **Backend is largely unpartitioned c-code, needs further breakdown to maximize advantages gained by using graphical data flow paradigm**
- **Waited for delivery of Mercury hardware and release of AIB for ~ one month**

Lesson Learned from Tool Usage

- **The GEDAE development environment greatly simplified capture and test of applications targeted for real-time distributed environments.**
  - **tools were easy to learn and use**
- **Concerns of team with "black box" effect of autocoding software turn out to be unfounded**
  - **visualization aids were very effective**
  - **insight to what the software was doing was actually better then some handcoded real-time software projects we have worked on**
- **The primary shortcomings of GEDAE are consistent with the level of maturity of the embedded capability which has been operational for less then a year.**

Tool Improvements

- **Tool improvements made during project development**
  - **Command program interface (AIB) added**
  - **Improvements to facilitate data flow and test**
    - **Automatic resizing of queues (Q) in GEDAE to minimize memory requirements and help isolate data flow problems**
  - **Improved and added primitives for V-array support especially for dynamic (variable consume and/or produce) and non-deterministic (variable threshold) boxes**
  - **System build (make) process improved**
- **Future improvements or additional features we would like to see**
  - **Better documentation of tool needed for embedding process, documentation for graph development on host is good**
  - **File management system is limited for large multi-user program development**
  - **More refined methods of shared data management to reduce amount of memory usage**
  - **Final documentation generator of graph, parameters, and command program interface would be very useful and save time**
  - **Summaries of trace, schedule and queue tables would be useful**

Module Outline

- Introduction
- Requirements Capture
- Fixed-Point Design - A RASSP Approach
- Simulation-Based Algorithm/Functional Design
- Network-Level DSP
- Link-Level DSP
- Signal Processing Simulators
- PGM/PGSE
- RASSP Software Generation
- **Summary**

.

Summary

- Introduced the algorithm and functional design environments for DSP
- Described the interaction between implementation and front end algorithm design
- Presented recent RASSP developments
  - Executable specifications
  - High-level test benches
  - Simulation-based design
  - Graphical and block-oriented algorithm capture
  - Examples of the above

References

[Arm95] Armstrong, J.R. , et al, "High Level Generation of VHDL Testbenches," *VHDL International Users Forum*, Spring 1995, pp 6.23 - 6.34.

[Army94] US Army Research Laboratory, The Documentation of Digital Electronic Systems with VHDL-The Army Handbook, Preliminary Final Draft Manuscript, February 9, 1994.

[Balaban84] Balaban P., et. al "Computer-Aided Modeling, Analysis, and Design of Communications Systems: Introduction and Issue Overview", *IEEE Journal on Selected Topics in Communications*, Jan 1984.

[Baudendistel93] Baudendistel K., Ph.D. Thesis, 1993, Georgia Institute of Technology, Atlanta, GA.

[BOSS86] Shanmugan K. S., Manning T.C. et. al, "Block-Oriented Systems Simulator (BOSS)", *IEEE , International Conference on Communications* , 1986, pp 36.1.1-36.1.10; © IEEE 1986

[Debardelaben95] Debardelaben J., Madisetti V., "HW/SW codesign for Signal Processing - A Survey and New Results," *Proceeding Asilomar Conference*, Oct. 1995.

[Egolf] Egolf T.W., Famorazdeh S., Madisetti V.K., "Fixed Point Codesign in DSP", VLSI Signal Processing VII Edited by Rabaey J., Chau P.M., Eldon J. Institute of Electrical and Electronics Engineers, Inc. New York, NY pg 113-126 ; © IEEE 1995

[Gajski95] Gajski D., Vahid F., "Specification and Design of Embedded Hardware-Software Systems," *IEEE Design & Test of Computers*, Spring 1995, pp. 53-67.

[Gupta93] Gupta R., G. De Micheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design & Test of Computers*, September 1993, pp. 29-41.

[Gupta94] Gupta R., Coelho C., De Micheli G., "Program Implementation Schemes for Hardware-Software Systems," *Computer*, January 1994, pp. 48-55.

References (Cont.)

[Kumar] Kumar S., Aylor J., Johnson B., Wulf W., "A Framework for Hardware/Software codesign," *Computer*, December 1993, pp. 39-45.

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

[Ismail95] Ismail T., Jerraya A., "Synthesis Steps and Design Models for codesign," *Computer*, February 1995, pp. 44-52.

[Jackman88] Jackman J. , Medeiros D.J., "A Graphical Methodology for Simulating Communication Networks", *IEEE Transactions on Communications*, April 1988; © IEEE 1988

[Kalavade93] Kalavade A., Lee E., "A Hardware-Software codesign Methodology for DSP Applications," *IEEE Design & Test of Computers*, September 1993, pp. 16-28.

[LMC95] Schamming B., *RASSP Methodology Working Group Meeting Software Process,* Lockheed Martin Corporation, March 16, 1995., Advanced Technology Lab, Camden, NJ

[LMC-ARCH] Lockheed Martin ATL RASSP Team, RASSP MYA Specification Volume I: Introduction to Model Year Architecture Version 1., Moorestown, NJ, September 1996.

[LMC-METH] Lockheed Martin ATL RASSP Team, *RASSP Methodology Version 2.0*, Moorestown, NJ, October 1995.[LMC-METH] Lockheed Martin ATL RASSP Team, *RASSP Methodology Version 2.0*, Moorestown, NJ, October 1995.

[LMC-Review] Lockheed Martin ATL RASSP Team, Final Review, Moorestown, NJ, April 1998.

[Madisetti94] Madisetti V., "Vive La Difference", *The RASSP Digest*, Vol. 1, No. 1, 4th Qtr. 1994, pp. 17-20.

[Messerschmitt84] Messerschmitt D.G., "A Tool for Structured Functional Simulation", *IEEE Journal on Selected Topics in Communications*, Jan 1984 ; © IEEE 1984

References (Cont.)

[MMC/LMC94] Management Communication Control, Inc., *Concept Definition Study Draft Technical Report,* prepared for Lockheed Martin Advance Technology Lab., May 1994. Used with permission.

[MYA95]]Martin Marietta Corporation, "RASSP Model Year Architecture Document", *Internal RASSP Document*, 1995.

[Myers95] Myers C., Dreiling R., "VHDL Modeling for Signal Processor Development," *Proceedings IEEE International Conference Acoustics, Speech, and Signal Processing*, May 1995.

[PGM90] Prepared by Naval Research Laboratory, *Processing Graph Method: Tutorial,* January 1990.

[Purvis91] Purvis M., "Hardware/Software codesign: A Perspective," *IEEE International Conference on Software Engineering*, 1991, pp. 344-351.

[RASSP94] Martin Marietta RASSP Team, *RASSP Methodology Version 1.0*, Moorestown, NJ, Martin Marietta Laboratories, December 1994.

[Richards97] Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997

[Shanmugan88] Shanmugan K. S., "An Update on Software Packages for Simulation of Communication Systems (Links)", *IEEE Journal on Selected Topics in Communications*, 1988, vol 6, pp 5-12 ; © IEEE 1988

[Shanmugan89] Shanmugan K. S., et. al. "Simulation Based CAAD Tools for Communication and Signal Processing Systems", *IEEE International Conference on Communications*, 1989, vol 3, pp 1454-1461.

References (Cont.)

[Shaw95] Shaw G., Anderson J., Madisetti V., "Assessing and Improving Current Practice in the Design of Application-Specific Signal Processors," *Proceeding IEEE International Conference Acoustics, Speech, and Signal Processing*, May 1995.

[Thomas93] Thomas D., Adams J., Schmit H., "A Model and Methodology for Hardware-Software codesign," *IEEE Design & Test of Computers*, September 1993, pp. 6-15.

[Vahey95] Vahey M., et al., "A Virtual Prototype VHDL Development Methodology," *VHDL International Users Forum*, Spring 1995, pp. 3.17-3.25.