



DSP Algorithm Design

RASSP Education & Facilitation Program

Module 23

Version 3.00

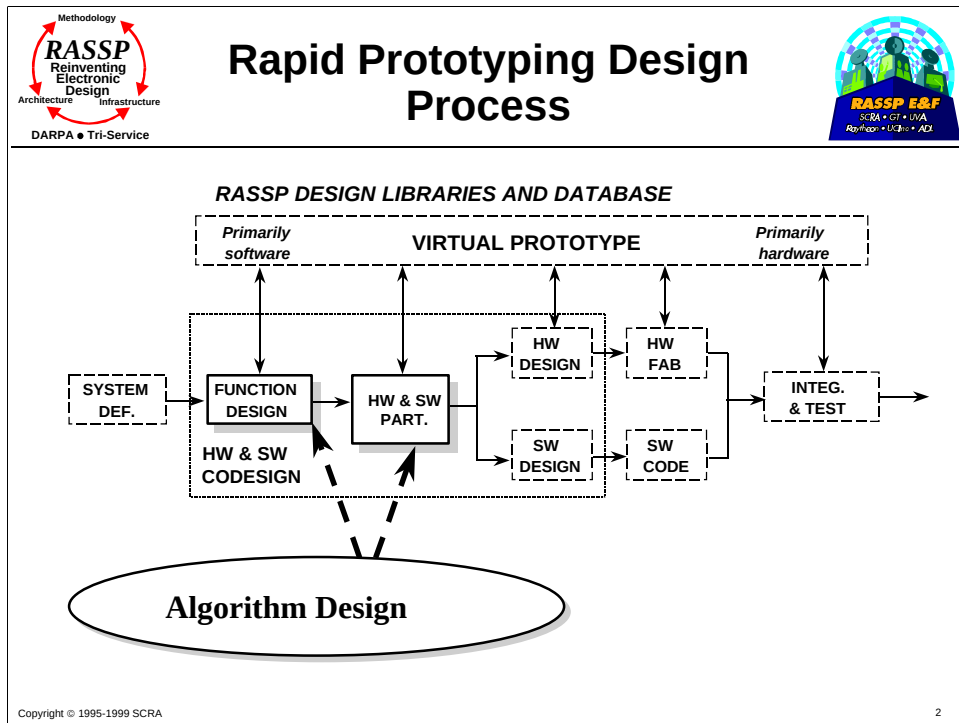
Copyright 1995-1999 SCRA

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

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

Copyright © 1995-1999 SCRA

1



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.



Module Goals



- | **State of the art survey in DSP algorithm design**
- | **Convey RASSP-related contributions and impact**
 - m **Learn how to unambiguously state the specifications of DSP systems for later verification of the final implementation**
 - m **Learn about the various levels at which a DSP system can be simulated**
 - m **Learn how to link the higher level algorithm design to lower levels**
 - m **Know more about the different CAD tools which can be used at the above levels to automate the task of algorithm development and simulation**
 - m **Learn partitioning and codesign of the hardware and software components of DSP systems.**

Copyright © 1995-1999 SCRA

3

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



Module Outline



- | **Introduction**
- | **Requirements Capture**
 - m **Basic approach**
 - m **Requirements capture**
 - q **SAR application**
 - q **IRST application**
 - q **Requirements capture tool RDD-100 overview**
 - m **Code generation**
 - q **Viterbi decoder application**
 - m **Data generation**
 - m **Testbench simulation and control**

Copyright © 1995-1999 SCRA

4

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.



Module Outline (Cont.)



- | **Fixed-Point Design - A RASSP Approach**
 - m **Motivation for fixed-point processing**
 - m **Fixed-point representations**
 - q **Simple fixed point**
 - q **Generalized fixed point**
 - q **Examples**
 - m **VHDL fixed point modeling**
 - q **Fixed-point packages**
 - q **High level modeling of processors**
 - q **QuickFix environment**
- | **Simulation-Based Algorithm/Functional Design**
- | **Network-Level DSP**

Copyright © 1995-1999 SCRA

5

Continuation of table of contents



Module Outline (Cont.)



- | **Link-Level DSP**
- | **Signal Processing Simulators**
- | **PGM/PGSE**
 - m PGM/PGSE overview
 - m Application development in PGM
 - m Issues in the use of PGM
- | **RASSP Software Generation**
- | **Summary**

Copyright © 1995-1999 SCRA

6

Continuation of table of contents



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

Copyright © 1995-1999 SCRA

7

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



Introduction



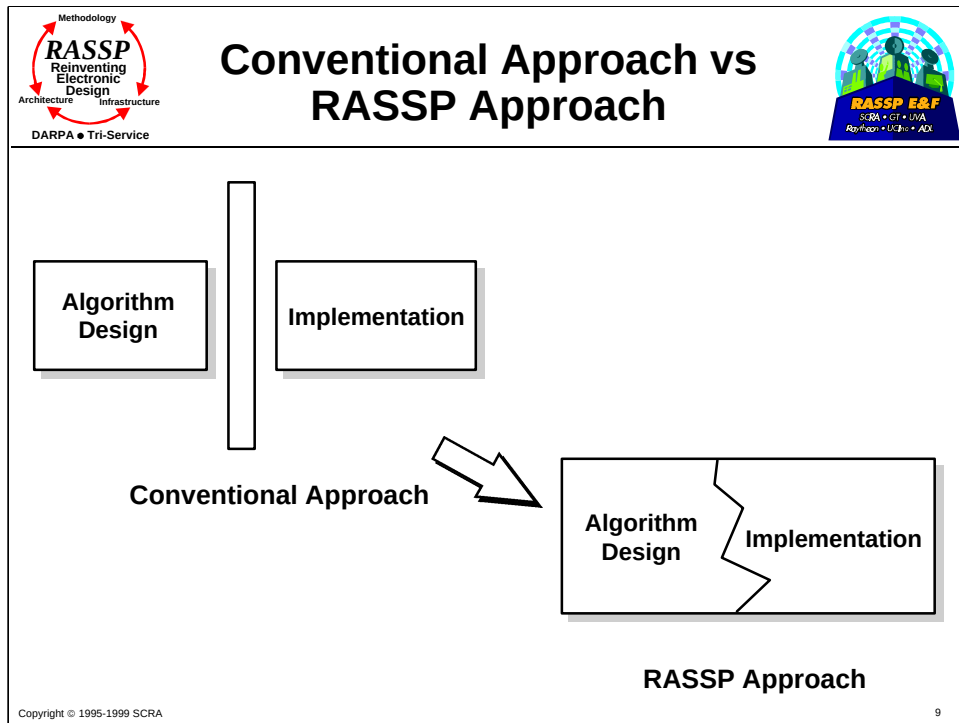
- | **The complexity of Signal Processing/Communication Systems algorithms has grown enormously**
- | **The time-to-market requirements have reduced considerably**
- | **There is a mismatch and dichotomy between algorithm development & design and algorithm implementation**

CONFLICTING REQUIREMENTS

Copyright © 1995-1999 SCRA

8

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.



Trends in Algorithm Design



- | New technologies are emerging
- | Need to integrate networking, transmission, sensors, data and signal processing
- | Mission requirements are changing and becoming flexible
- | Numerical and storage requirements are growing
- | There is often inadequate information on nature of data
- | There is little reuse of part designs (beyond organizational barriers)
 - m Approach has been to REUSE the designer
- | There is no automation at higher levels of abstraction
- | Work is done by multi-person teams of designers and implementation experts

Copyright © 1995-1999 SCRA

10

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



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

Requirements of RASSP Algorithm Development Environment



RASSP E&F
SCRA • GT • USA
Rapidgo • UCI • ADX

- | **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)**

Copyright © 1995-1999 SCRA

11

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



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

Copyright © 1995-1999 SCRA

12

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.



Section Outline



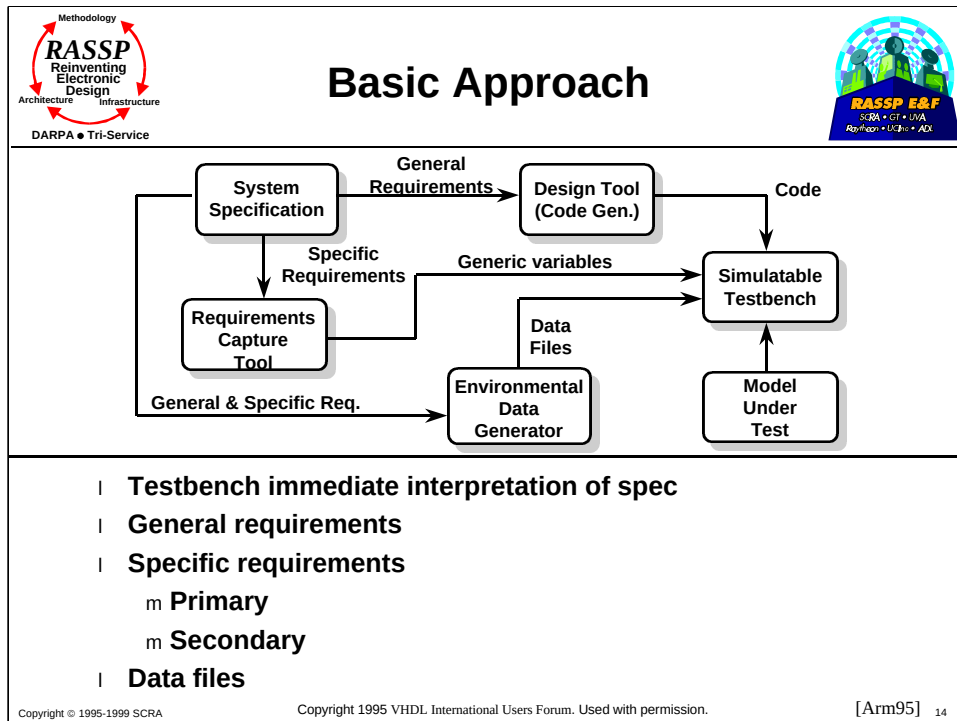
- | **Requirements Capture**
 - m **Basic approach**
 - m **Requirements capture**
 - m **Code generation**
 - m **Data generation**
 - m **Testbench simulation and control**

Copyright © 1995-1999 SCRA

13

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.

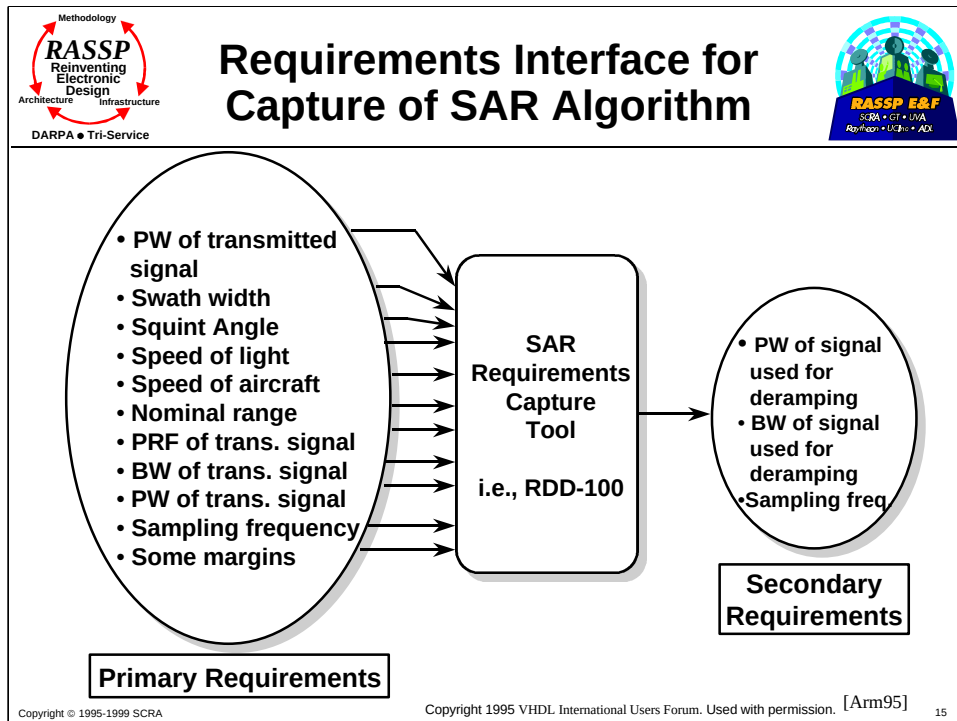
The above figure is a high-level testbench generation system to implement this strategy.

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.

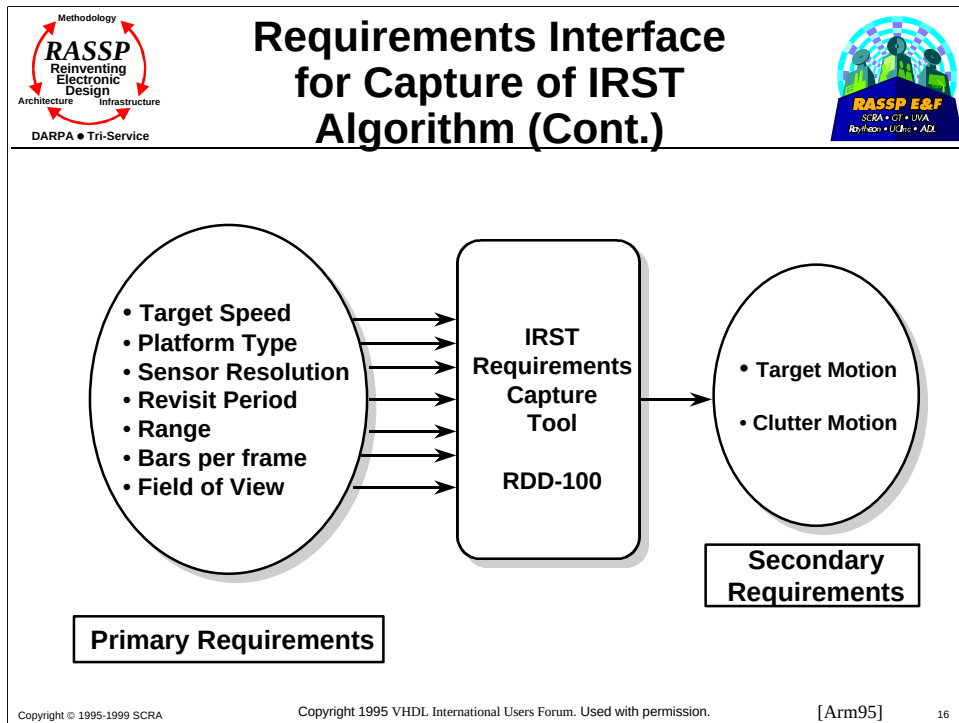
The secondary requirements are derived from the primary requirements through use of the mathematical modeling capabilities of the capture tool.

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.



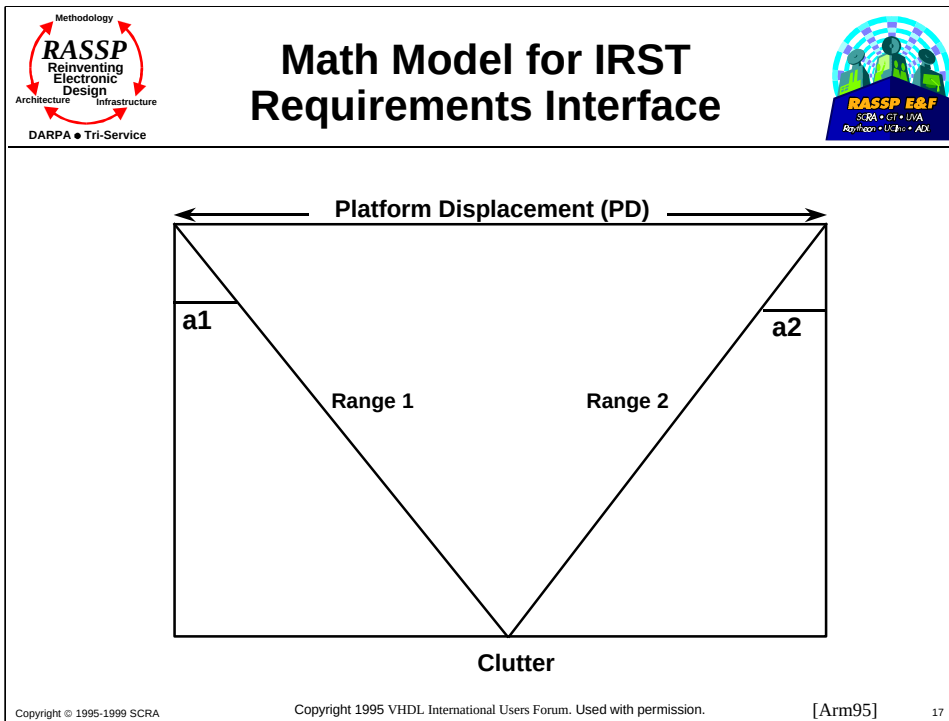
This is a similar diagram used for the IRST algorithm. The primary and secondary requirements are listed. Their derivation is on the next slide's note page.

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. = $a1 + a2$

$\sin(a1) = (.5 \cdot PD) / \text{range } 1$

$a1 + a2 = 2 \cdot \arcsin(PD / (2 \cdot \text{range } 1))$

$PD = (a1 + a2) / \text{PAFOV}$

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 = $1609.344 \cdot \text{range}$

Platform Disp. = Platform velocity * Revisit period

Clutter motion = $(2 \cdot \arcsin(\text{platform disp.} / (2 \cdot \text{clutter period})) / \text{sensor res. factor}$

Target range = $1609.344 \cdot \text{range}$

Target velocity = Mach to m/s * target range

Target disp. = Target velocity * revisit period

Target motion = $(2 \cdot \arcsin(\text{target disp.} / (2 \cdot \text{target range})) / \text{sensor res. factor}$



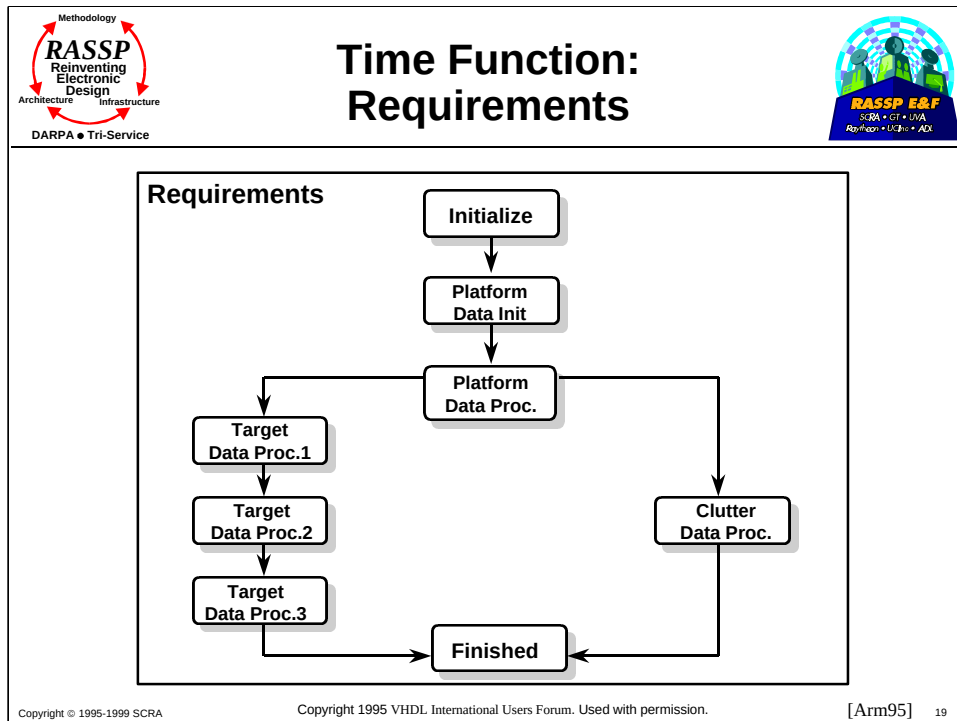
RDD-100: Definition Overview

- | **Behavior diagram**
 - m Graphical representation of behavior of functions in terms of time-flow sequence
- | **Function**
 - m Part of system that carries out an action, usually converting an input to an output
- | **Item**
 - m Something a function accepts or produces (i.e., I/O)
- | **Discrete function**
 - m Function at lowest level of detail w/o subfunctions
- | **Time function**
 - m Higher-level function which can have discrete functions and/or time functions as sub-functions

Copyright © 1995-1999 SCRA 18

[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.

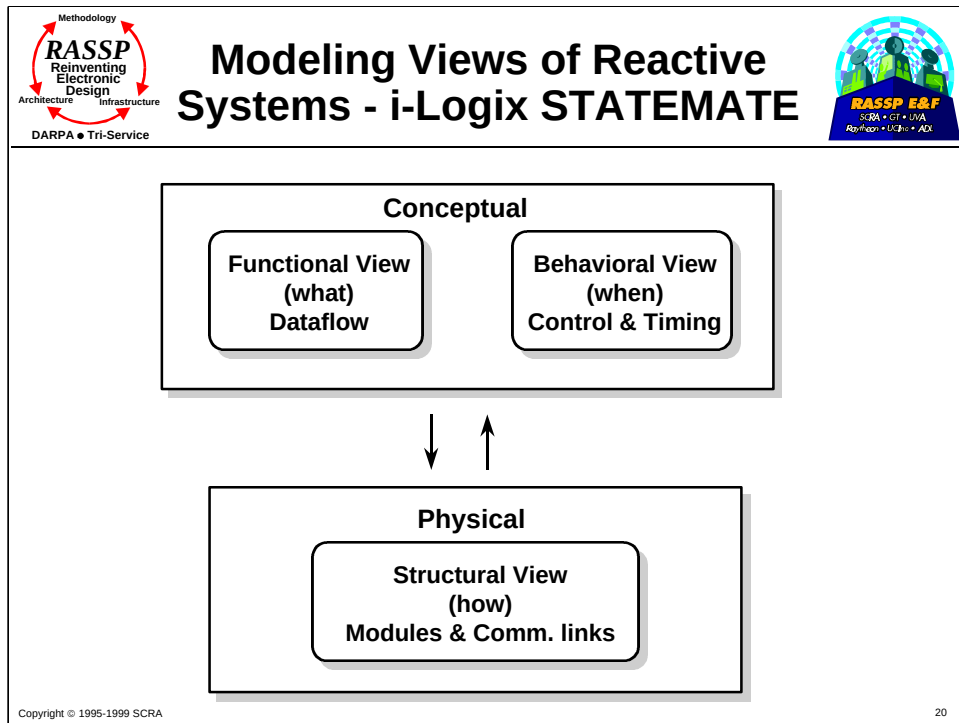


The highest-level time function developed was Requirements. It was decomposed into discrete functions: Initialize, Platform Data Init., Clutter Data Proc., Target Data Proc 1, Target Data Proc 2, and Target Data Proc 3. The inputs, outputs, and intermediate values were stored as discrete items.

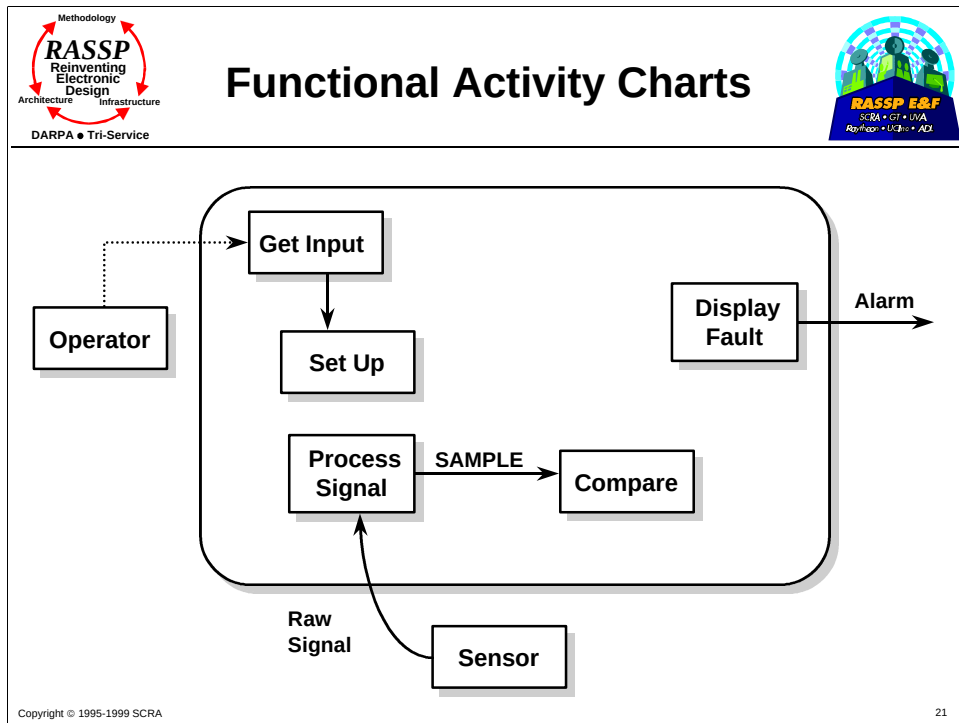
A description of each of discrete function follows:

- | 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.

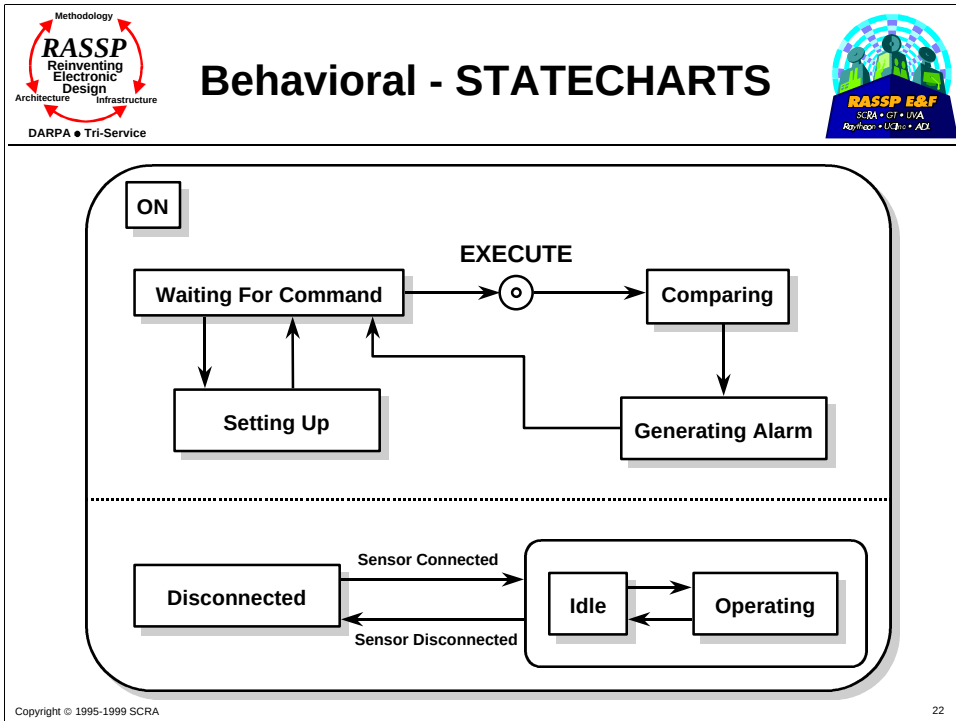
The Requirements time function was verified using the Dynamic Verification Facility of RDD-100. The primary requirements were assigned to their respective items and the secondary requirements were observed.



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).

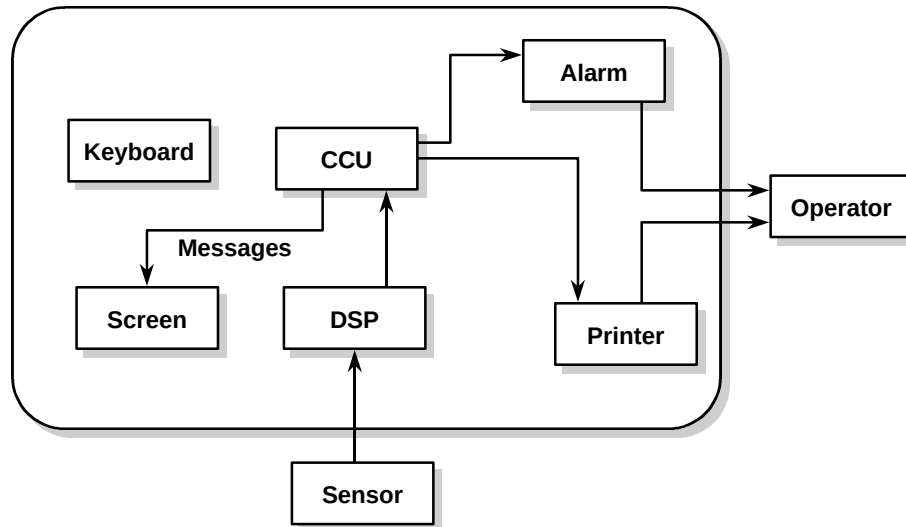


Example of a functional activity chart in i-Logix for describing the list of functions and dataflow.

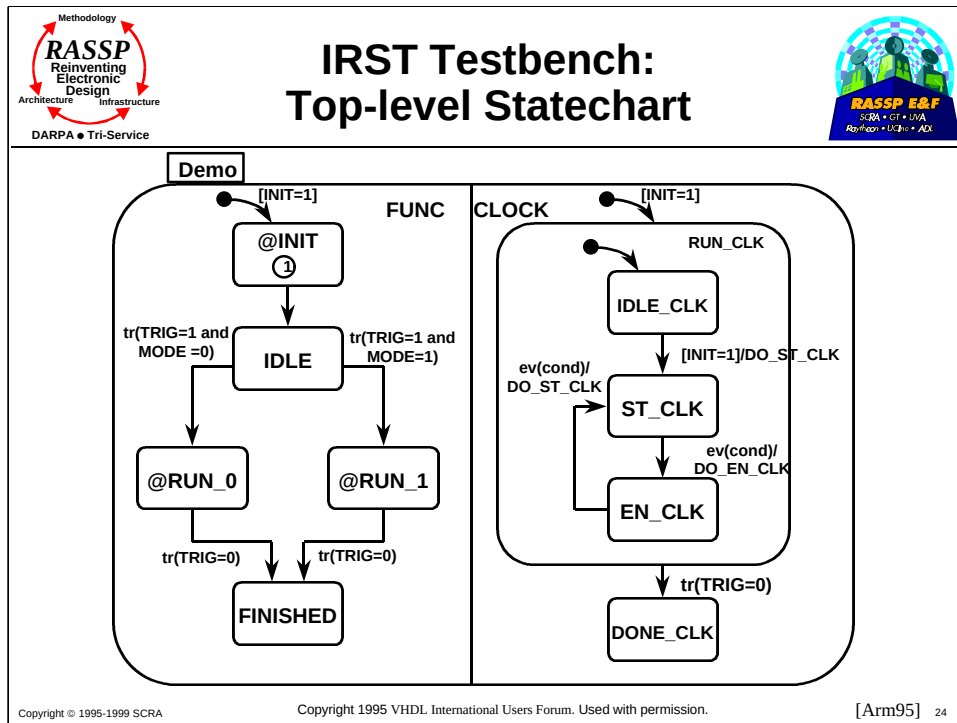


Example of the timing behavior of an alarm system captured as part of i-Logix. Note: concurrency in activities.

Structural - Module Charts



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

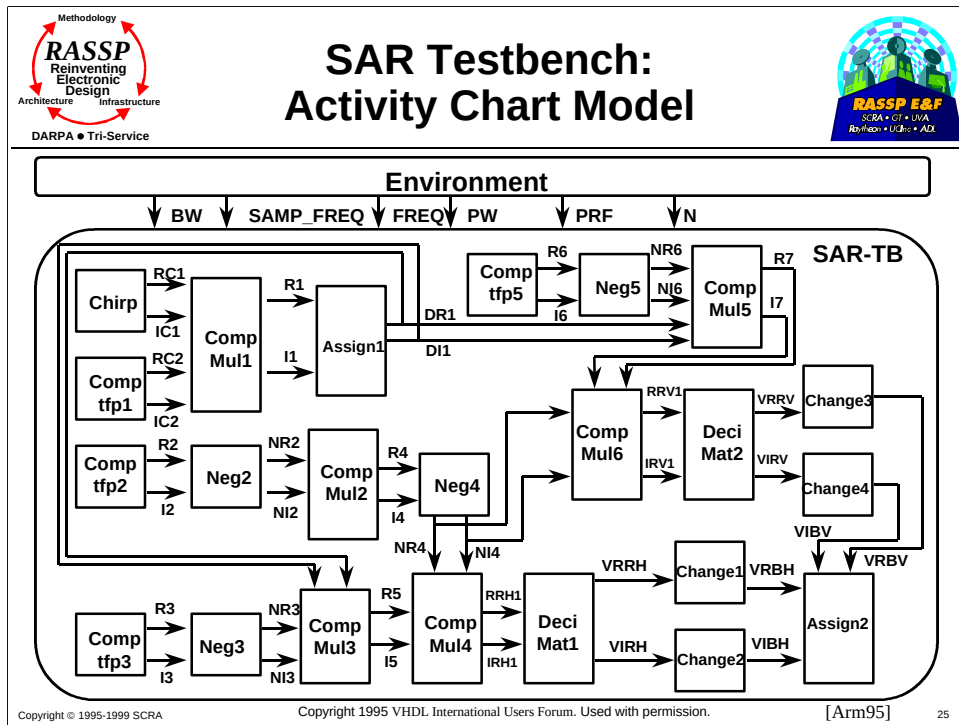


Statecharts are modified versions of state transition diagrams. They are made up of states and transition from one state to another. A label event(cond)/action is placed on each transition arrow describing the condition for transition and action to be taken.

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.



RASSP
Reinventing
Electronic
Design
Infrastructure
DARPA • Tri-Service

Code Generation



RASSP E&F
SCRA • GT • USA
Rapidgo • U.S. • ADX

I Two main approaches

- m Behavioral testbench development**
 - q Uses CASE tools to develop high-level models (i.e., i-Logix Express VHDL)**
 - q Statecharts and activity charts describe behavior**
 - q Advantages include**
 - í Reduced test time required for large models**
 - í Testbenches can be used for a class of systems which have same general functionality**
 - í No detailed internal structure required**
- m Structural testbench development**
 - q Uses schematic capture tools to construct the testbench from library of primitives**
 - q Advantage**
 - í Uses existing commercial schematic capture tools**

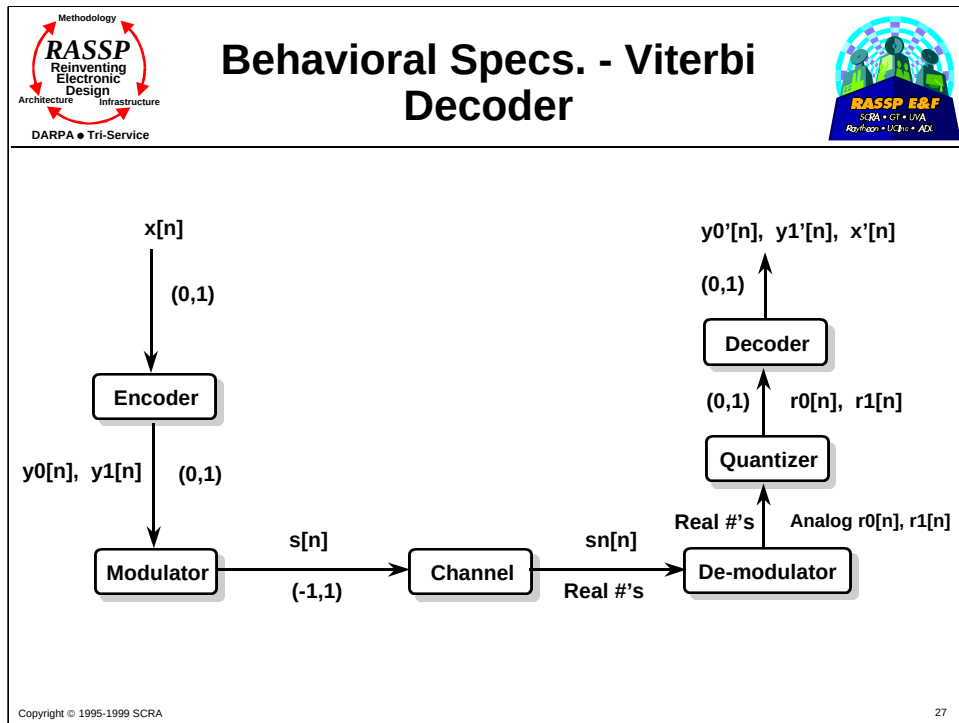
Copyright © 1995-1999 SCRA
26

[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.

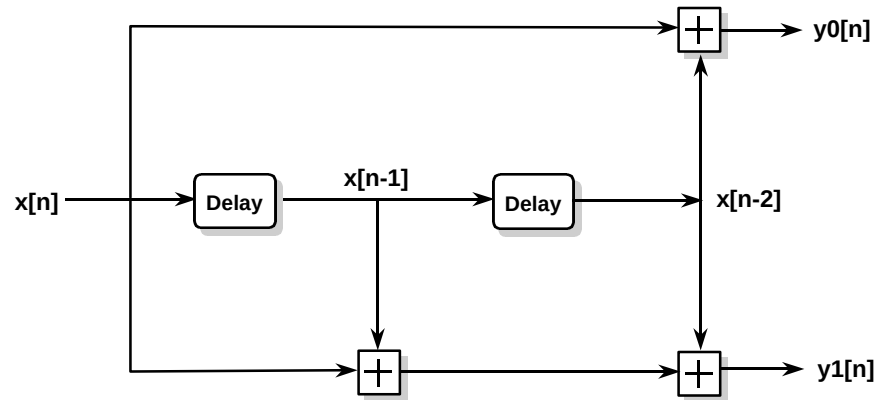
The CASE tool used for the first approach was i-Logix's Express VHDL which uses Statecharts to describe control flow and activity charts to describe data flow.



Let us consider a communication system. The input signal $x[n]$ is encoded using a rate 1/2 linear convolutional code. The encoder outputs y_0 and y_1 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.

Linear Convolutional Encoder



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

VHDL Code for the Encoder

```
ARCHITECTURE fsmd OF encoder IS
BEGIN
    PROCESS (clock)
        VARIABLE state : integer RANGE 0 to 3 := 0;
    BEGIN
        IF rising edge of the clock THEN
            CASE state IS
                Set the next state and the two outputs
                depending on the present state and the
                input
            END CASE;
        END IF;
    END PROCESS;
END fsmd;
```

User defined variables are
underlined. Key words are
in capitals.

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

BPSK Modulator

- | **Convert parallel output of encoder to serial**

- m Send $y0[n]$ at the rising clock edge
- m Send $y1[n]$ at the falling clock edge

- | **Transmit**

- m **-1** to represent a 0
- m **+1** to represent a 1

Another example of a commonly used communications subsystem.

VHDL for the Modulator

```
ARCHITECTURE behavior OF bpsk IS
BEGIN
    PROCESS (clock)
    BEGIN
        IF rising edge of the clock THEN
            If first input = 1 then transmit a +1
            if it is 0 then transmit a -1
        ELSIF falling edge of the clock THEN
            If second input = 1 then transmit a +1
            if it is 0 then transmit a -1
        END PROCESS;
    END behavior;
```

VHDL executable requirements of the same BPSK modulator.

AWGN Channel

| Signal to Noise Ratio (SNR) and Variance

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

$$\text{Variance} = \sigma^2 = N_0$$

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

| Gaussian Noise Generation

$$x_1 = \exp(-(y_1^2 + y_2^2)/2)$$

$$x_2 = (\arctan(y_2/y_1)) / (2\pi)$$

- m Choose x_1 and x_2 to be Uniform random variables over $(-1,1)$
- m Then y_1 and y_2 are Gaussian random variables with mean 0 and variance 1
- m Scale y_1 and y_2 to get the right variance

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

VHDL for the Channel

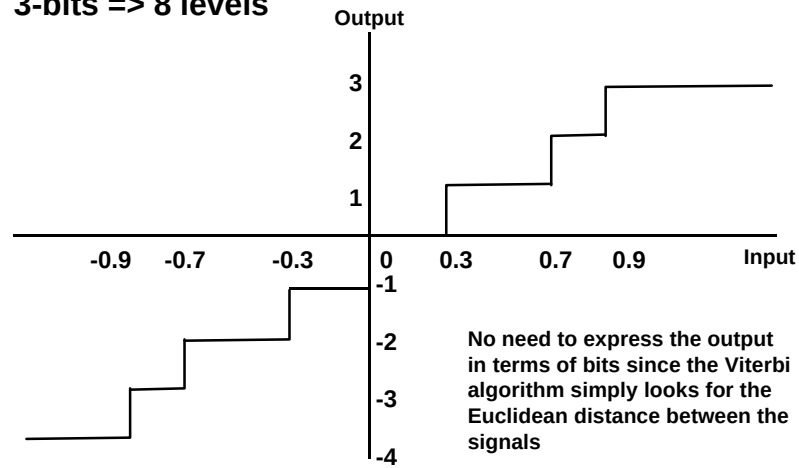
```
ARCHITECTURE behavior OF channel IS
BEGIN
    PROCESS (input)
    BEGIN
        sigma := 10**(-snr/20.0);
        gaussian(seed1,seed2,temp);
        noise := sigma * temp;
        output <= REAL (input) + noise;
    END PROCESS;
END behavior;
```

gaussian is a function defined by the programmer in a library

Representation of the channel in VHDL (executable requirements)

3-bit Quantizer

3-bits => 8 levels



A quantizer is represented in graphical form.

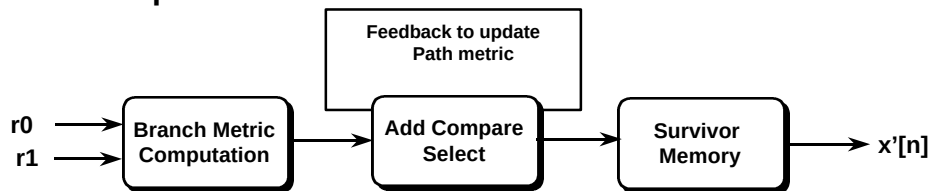
VHDL for the Quantizer

```
ARCHITECTURE behavior OF quantizer IS
BEGIN
    PROCESS (clock)
    BEGIN
        IF event on the clock THEN
            Depending on the range in which the
            input lies decide the output
            IF rising edge of the clock THEN
                actually send the output
            IF falling edge of the clock THEN
                update the internal registers
            END IF;
        END PROCESS;
    END behavior;
```

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

Viterbi Decoder Using the Register Exchange Method

- | **Maximum Likelihood Sequence Estimation (MLSE)**
- | **Most efficient decoder for convolutional codes**
- | **Applications**
 - m High performance equalizer for mobile communication
 - m TCM decoder for V.32 modem
- | **Implementation Structure**



Another commonly used communication subsystem that uses the Viterbi decoder.

VHDL for the Viterbi Decoder

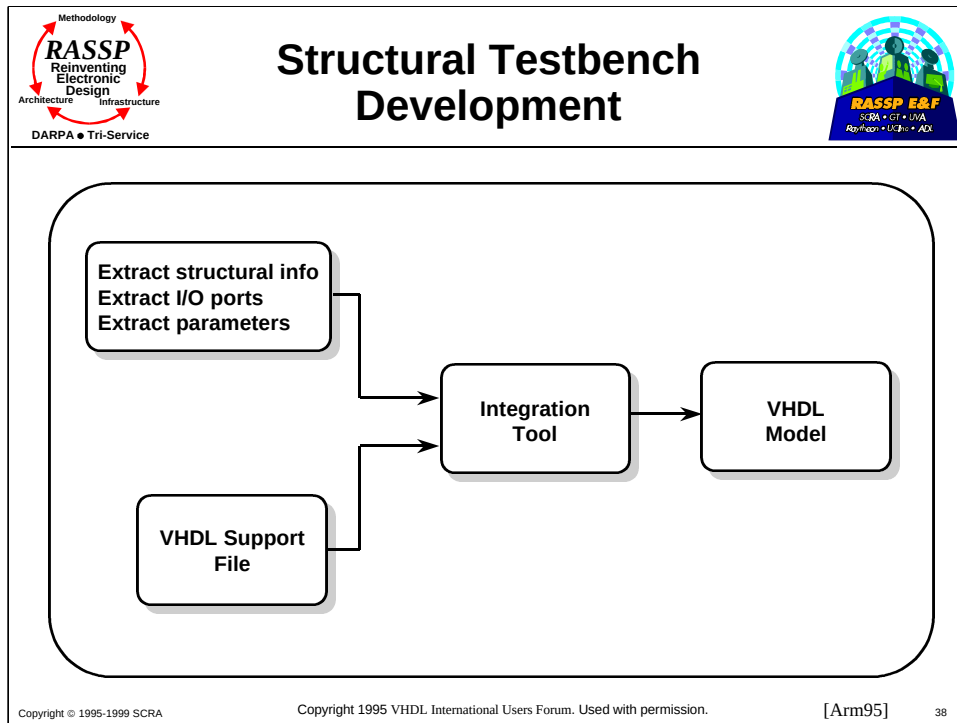
```

ARCHITECTURE behavior OF quantizer IS
BEGIN
    PROCESS (clock)
    BEGIN
        Delay activation of the decoder till the first sample
        reaches it
        IF rising edge of the clock and THEN
            Calculate branch metrics using inner
            products between the received word and
            the code word
            Compare and select the surviving path
            Exchange register contents
        END IF;
    END PROCESS;
END behavior;

```

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.

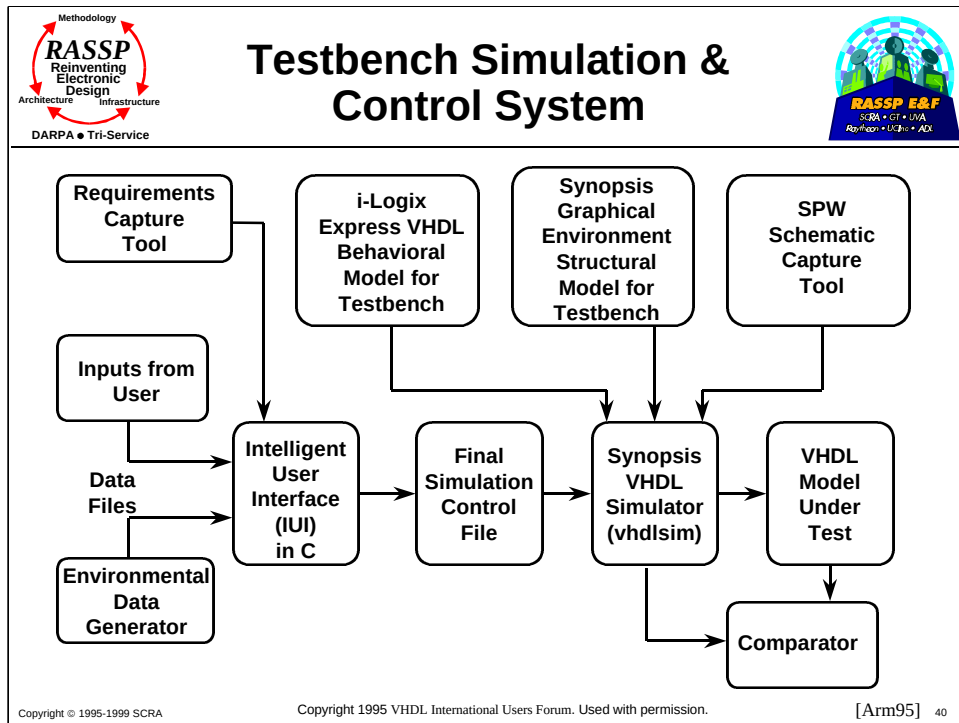
The Integration tool is used to integrate all the structural information, I/O information, and parameters captured by shell scripts. It also uses a VHDL support file which is a package of library VHDL procedures corresponding to SPW primitives.

Data Generation

- | **Comdisco/Cadence SPW, xpatch, or Matlab used to generated data files for model under test**
 - m **SPW** uses “sink” primitive to dump to file in ascii format
 - m **xpatch** used to generate radar sensor data files
 - m **Matlab** used to generate clutter file data

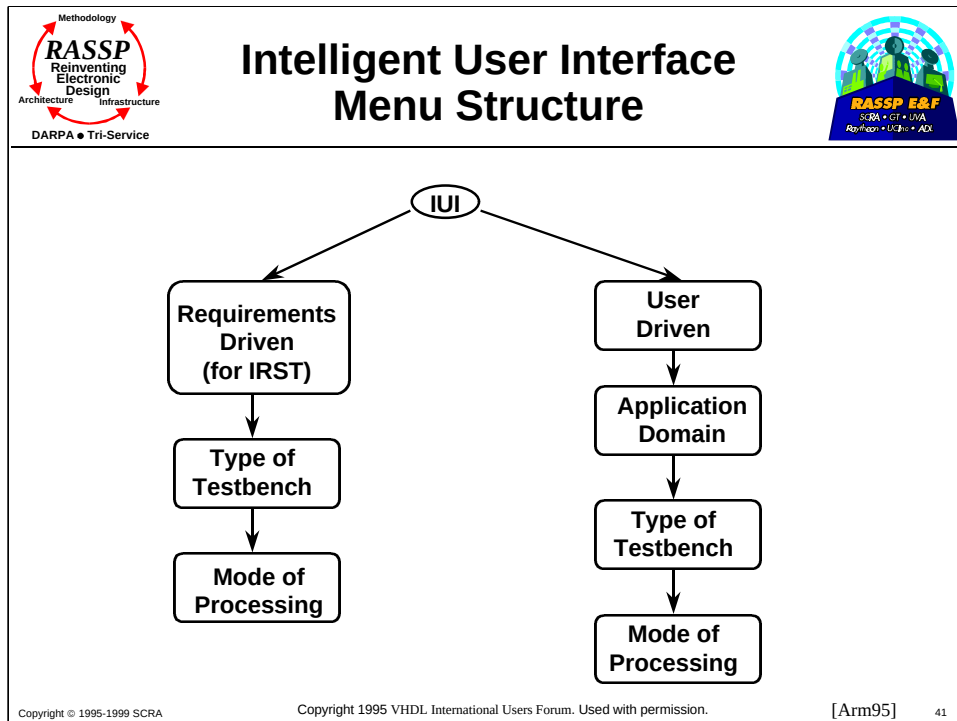
[Arm95]

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.



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

Copyright © 1995-1999 SCRA

42

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.

Section Outline

| **Fixed-Point Design**

- m **Motivation for fixed-point processing**
- m **Fixed-point representations**
 - q **Simple fixed point**
 - q **Generalized fixed point**
 - q **Examples**
- m **VHDL fixed point modeling**
 - q **Fixed-point packages**
 - q **High level modeling of processors**
 - q **QuickFix 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.

Motivation for Fixed Point

- | **Lower cost implementation and design complexity**
- | **Less power**
- | **Smaller area**
- | **Processor examples**
 - m **Fixed-point processors**
 - q **Motorola 56000, Power: 90 mA @ 5 V**
 - q **Texas Instruments TMS320C50, Power: 60 mA @ 5V**
 - q **Analog Devices 2100, Power: 60 mA @ 5V**
 - m **Floating-point processors**
 - q **AT&T DSP3210, Power: 220 mA @ 5V**
 - q **Texas Instruments TMS320C30, Power: 300 mA @ 5V**

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.

Fixed-point Representations

- I **Simple fixed point**
 - m Field of bits with radix point on bit boundary
 - m Power of 2 scaling
 - m Notation: fix<x.y>
 - m Integer form: Radix point to right of LSB
 - m Fractional form: Radix point one bit to right of MSB (sign bit)
- I **Generalized fixed point**
 - m Radix point not necessarily on bit boundary
 - m Integral (exact) and fractional (approximate) forms
 - m Notation: Integral form
 - q $i_x \Delta_x$ where i_x is the integral field and Δ_x is the stepsize
 - m Notation: Fractional form
 - q $\alpha_x \phi_x$ where α_x is the fractional field and ϕ_x is the fieldsize

[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.

Fixed-point Examples

I Simple fixed point

m Given the following simple fixed-point type: `fix<3.2>`

3.75 |--> 011.11

I Generalized fixed point

m Integral form: Given a precision of 5 bits and a stepsize of 1/5, then the number 2 maps to 01010

2 |--> 01010

m Fractional form: Given a precision of 8 bits and a fieldsize of -10 to 10, then the number 1/7 maps to 00000001 or 00000010 based on the rounding mode

1/7 |--> 00000001 or 00000010

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.



RASSP
Reinventing
Electronic
Design
Architecture Infrastructure
DARPA • Tri-Service

VHDL Fixed-point: High-level Modeling



RASSP E&F
SCRA • GT • USA
Rapidgo • U.S. • ADX

- | **VHDL has two basic data types (scalar and composite)**
 - m **Numeric (Integer and Real)**
 - m **Enumeration (boolean, bit, and character)**
 - m **Physical (time)**
- | **Fixed-point packages to perform specific calculations**
 - m **Overloaded operators**

```

package BIT_16pack is
  type FIX16 is
    record
      field : real;
      frac  : integer;
    end record
  function "+" (op1 : FIX16;
               op2 : FIX16) return FIX16;

  function "*" (op1 : FIX16;
               op2 : FIX16) return FIX16;

  function "-" (op1 : FIX16;
               op2 : FIX16) return FIX16;

  function "/" (op1 : FIX16;
               op2 : FIX16) return FIX16;
end BIT_16pack;

```

Copyright © 1995-1999 SCRA
© IEEE 1995
[Egolf] 47

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 than, negation, and not equal to.



VHDL Processor-specific High-level Modeling



- | **Package expresses the functionality of the processor to be modeled**
- | **Important functionality**
 - m **Adder and multiplier input and output widths**
 - m **Accumulator sizes**
 - m **Shifter functionality**
 - m **Rounding/Truncation and overflow properties**

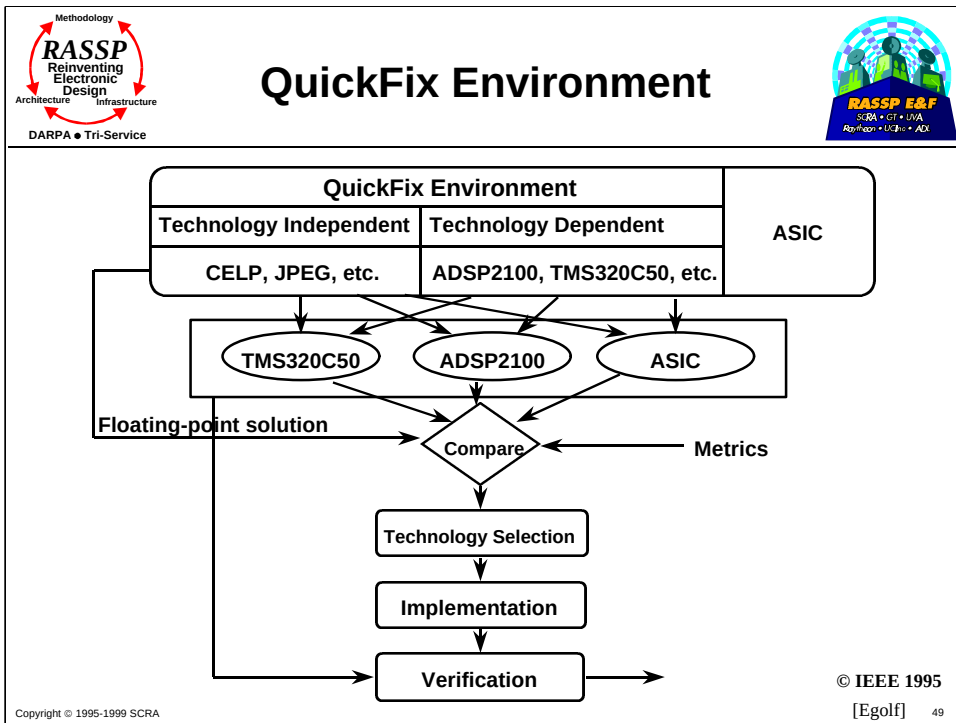
```

package ADSP2100pack is
function "+" (op1 : BIT_16;
              op2 : BIT_16) return BIT_16;
function MULT (op1 : integer;
               op2 : integer;
               mode : string) return BIT_40;
function RND (op1 : BIT_32) return integer;
function TRUNC (op1 : BIT_32) return integer;
function ADD_40 (ad1 : BIT_40;
                 ad2 : BIT_40) return BIT_40;
function MAC (accum : BIT_40;
              mp1 : integer;
              mp2 : integer;
              mode : string) return BIT_40;
function GET_EXP (op1 : BIT_VECTOR)
  return integer;
function NORMALIZE (op1 : BIT_VECTOR;
                    shift : integer);
  return BIT_32;
end ADSP2100pack;

```

Copyright © 1995-1999 SCRA
© IEEE 1995
[Egolf] 48

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.

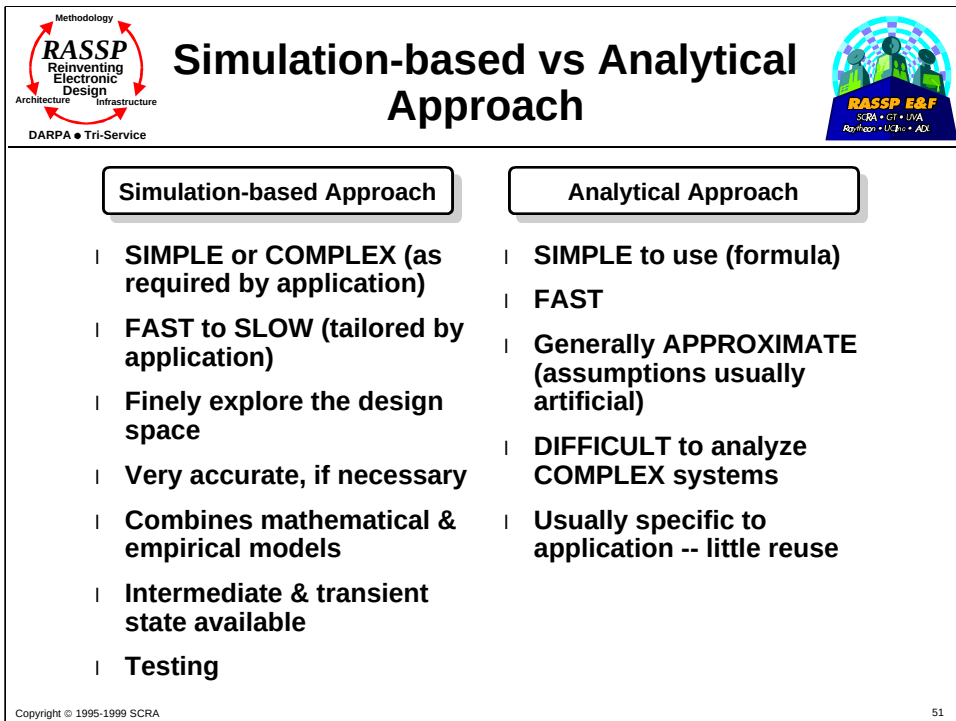
This can then point the designer to the correct architecture to use to implement the algorithm's behavior.



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.



Trends in Simulations/Tools and Environments



- | **1970s - Individual programs**
 - m Batch mode execution on mainframes
 - m Outputs consist of tables and numbers
 - m Considerable time on debugging and less time on application
- | **1980s - Simulation environments and languages**
 - m Model libraries
 - m Interoperability
 - m Computer-aided design and analysis
 - m Better user interfaces
 - m Distributed and parallel environments
 - m Advanced database management

Copyright © 1995-1999 SCRA

52

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



Trends in Simulations/Tools and Environments (Cont.)

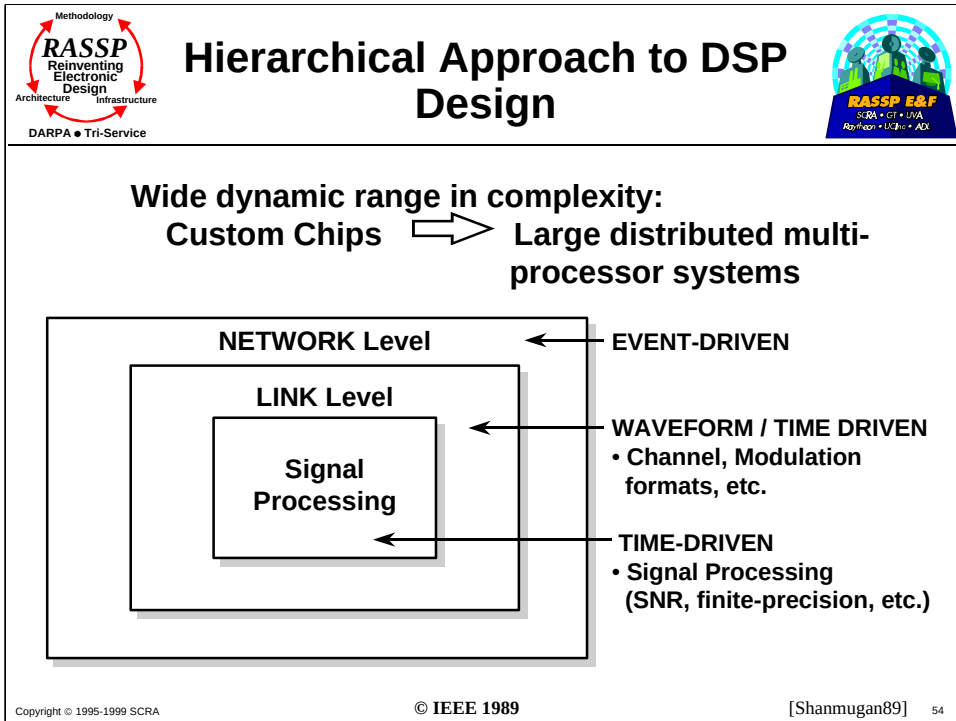


- | **1990s - Managing simulation complexity, interoperability, and reuse (little programming required, if any)**
 - m **Hierarchical simulation and Block-oriented**
 - m **Mixed domains of computation and interaction**
 - m **Larger application-specific focus**
 - m **Simulation and verification**
 - m **Automation**
 - m **Links to lower levels of abstraction**

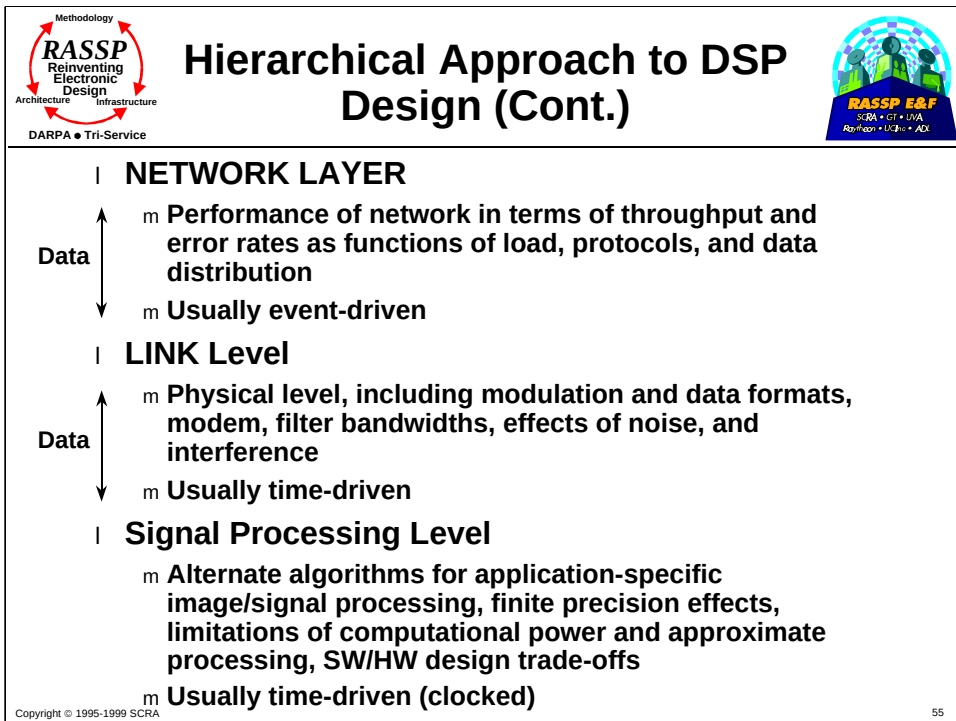
Copyright © 1995-1999 SCRA

53

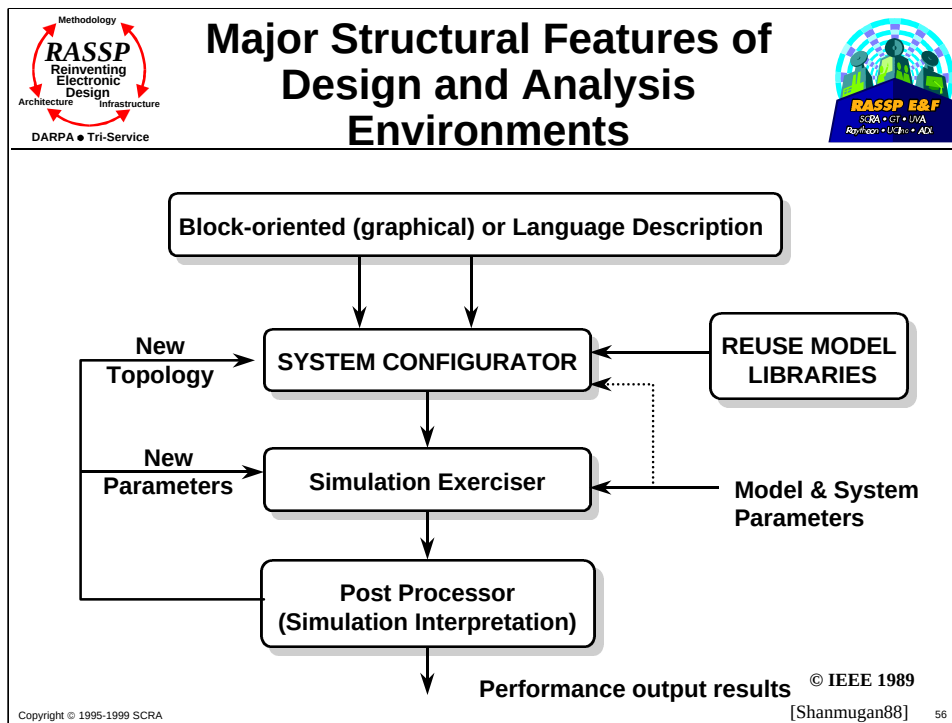
The trends for the 90's are listed above. The Unified Modeling Language (UML) is a late 90s evolution that promises to unify the advantages of various modeling paradigms.



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

- m Selects functional blocks from model library
- m Connects them in the desired topology
- m Can set parameters

| Simulation Exerciser

- m Sets up the execution of the simulation
- m Schedules, assigns, and allocates resources
- m Generates and uses testbenches
- m Performs checkpointing and storage of simulation state
- m Performs garbage collection and recovery

| Post Processor

- m Examines time & event histories
- m Computes performance metrics
- m Formats and displays interpreted results
- m Decides on the termination conditions (iterations)

Copyright © 1995-1999 SCRA

57

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



Simulation Inputs (Graphical and Language)



- | **Preferred approach - functional blocks/package to be written in C & FORTRAN**
 - m Do not allow input via block diagram (engineer's approach)
- | **Special-purpose Simulation languages that are portable across various computing platforms**
 - m E.g.: SIMSCRIPT, GPSS, SLAM, SIMULA, VHDL
- | **Graphical Languages such as:**
 - m Statemate/Statechart
 - m PGM/PGSE environments
 - m Ptolemy
 - m Blossim
 - m Comdisco (SPW, BOSS, BONES)
 - m COSSAP

Copyright © 1995-1999 SCRA

58

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



Features Required from an Input Language



- | **Training Required**
 - m Ease of learning the language
 - m Ease of conceptualizing design problems
 - m Ease of programming
- | **Coding Considerations**
 - m Integration
 - m Degree to which code is self-documenting
- | **Portability**
 - m Language availability on various platforms
- | **Flexibility**
 - m Supports various modeling domains
 - m Statistics/metric calculation capabilities
 - m List/queue processing capabilities

Copyright © 1995-1999 SCRA

59

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



Features Required from an Input Language (Cont.)



| Processing Considerations

- m Ease of producing reports
- m Ease of user-interface usage

| Debugging Capabilities

- m Ease of debugging
- m Reliability and efficiency of compilers

| Run-time Considerations

- m Execution speed
- m Memory management
- m Concurrency extraction

Copyright © 1995-1999 SCRA

60

Features of languages (Contd.)



Other Functional Components



| Model Library

- m Contains large number of validated models of functional behavior
- m Should be capable of being expanded by user
- m Provides inter-operability and standardized interfaces

| Model Builder

- m Selects building blocks
- m Sets parameter values
- m Operates at schematic capture-level
- m Is hierarchical
- m Eliminates programming by user

| Simulation Manager

- m Translates model to executable program
- m Links to libraries
- m Assigns, schedules, and allocates resources
- m Performs database management

Free the Algorithm
Developer from
burdensome tasks

Copyright © 1995-1999 SCRA

61

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



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

Copyright © 1995-1999 SCRA

62

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



Network-level CAD Tools

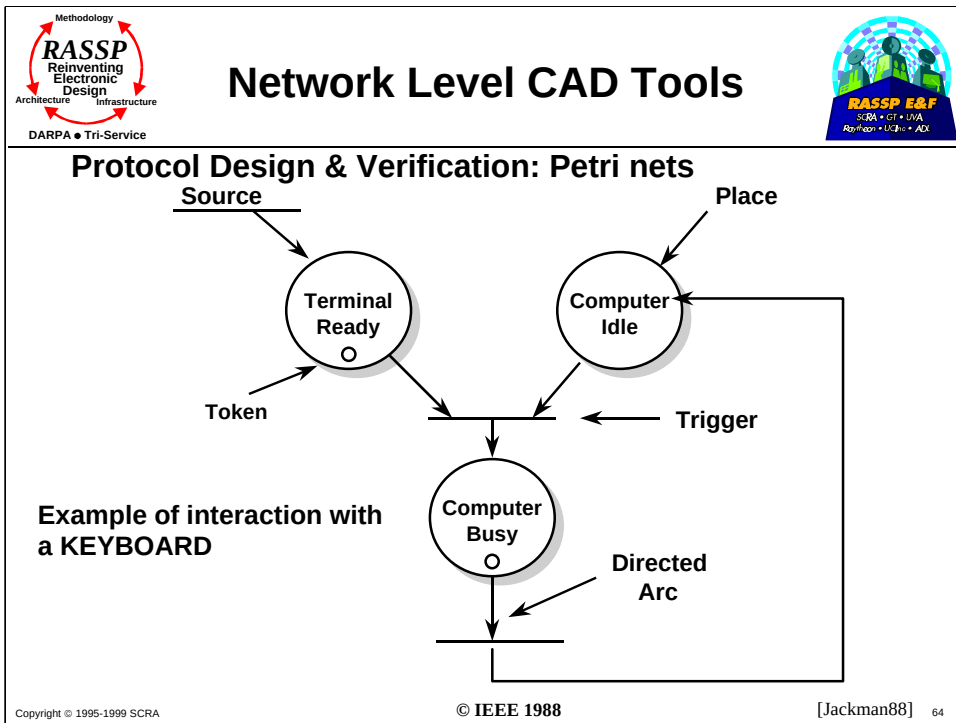


- | **Analytic tools**
- | **Simulation tools - most powerful**
- | **Emulation tools & testbeds (usually for testing protocols)**
- | **Focus on a NETWORK-LEVEL simulator**
 - m Finite state machines and Petri nets
 - m Queuing theory
 - m Collection of interacting procedures (SIMULA, GPSS)
- | **Network design and modeling consists of three major tasks**
 - m Protocol Design and Test - majority of the work
 - m Topology (physical)
 - m Processing nodes (attributes)

Copyright © 1995-1999 SCRA

63

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


Copyright © 1995-1999 SCRA
© IEEE 1988
[Jackman88] 64

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



RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA • Tri-Service

Petri Nets



RASSP E&F
SCRA • CT • USA
Reprogram • U.S. • ADX

| Petri nets have NO time delays and can be used for protocol verification only

m We need modified Petri nets (used in most simulation environments, e.g., BONES)

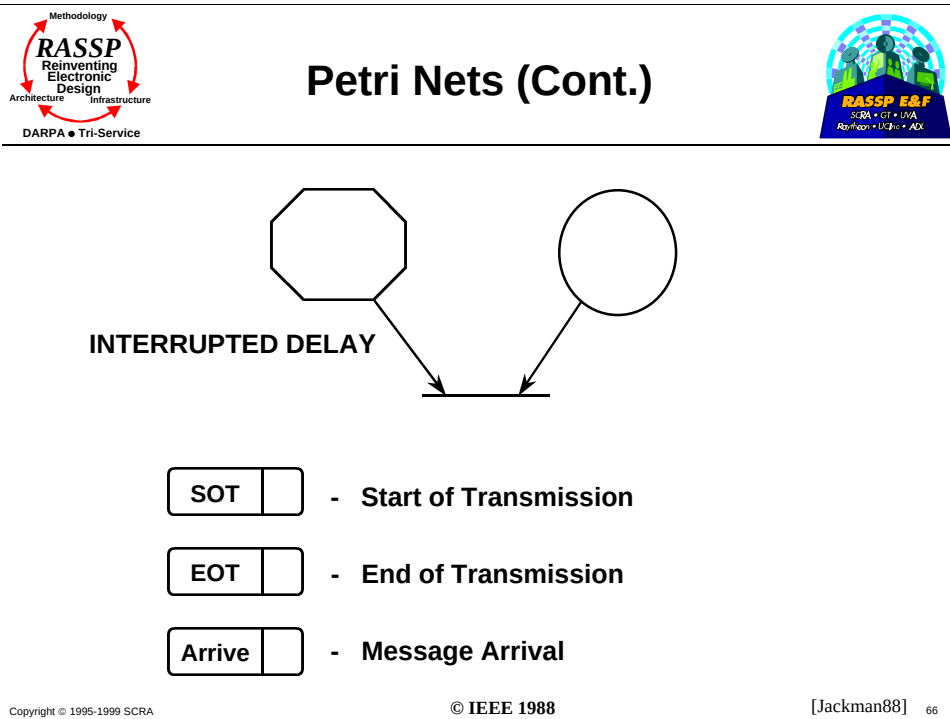
MODIFIED PETRI NET	STANDARD PETRI NET
Regular Place 	Place 
Delay Place 	Trigger 
Link Place 	Directed Arc 
Trigger 	Token 
Directed Arc 	
Not Edge 	
Primary Edge 	
Token 	

Copyright © 1995-1999 SCRA

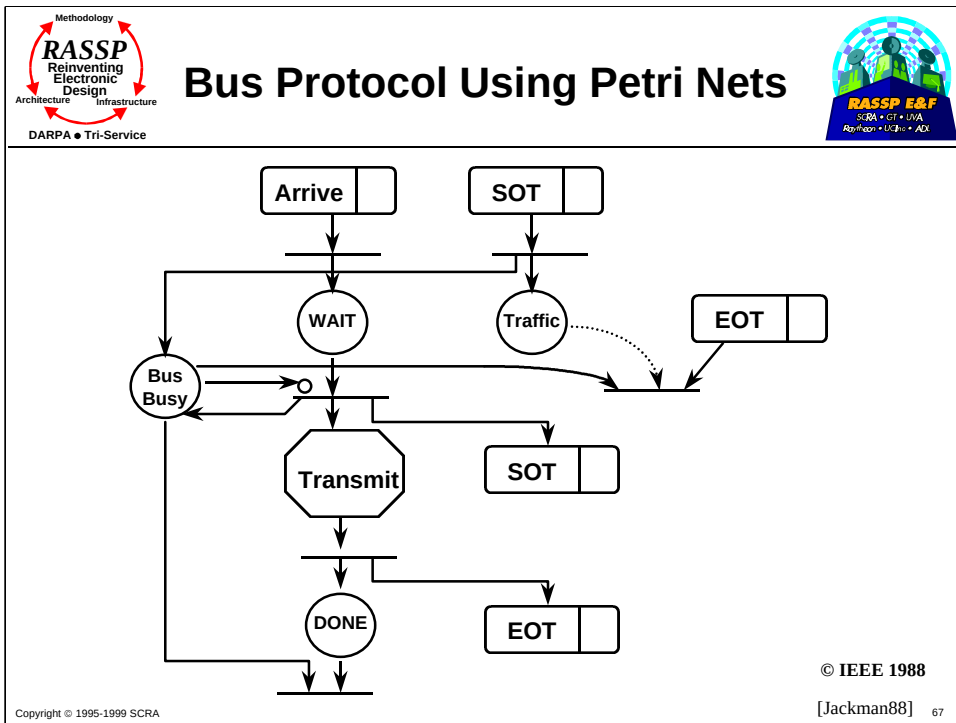
© IEEE 1988
[Jackman88] 65

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.

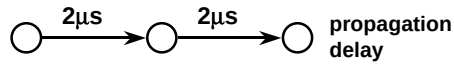


Example of the use of modified Petri nets in timing simulation in addition to protocol verification.

Detailed Example

I Topology

- m Physical connections
- m Propagation velocity of the medium
- m Transmission rate of the medium
- m Transmission type



I Node

- m Arrival distribution of packets
- m Packet-length distribution
- m Location of node on the network
- m Traffic pattern

I Protocol

- m Routing table for source/sink configuration

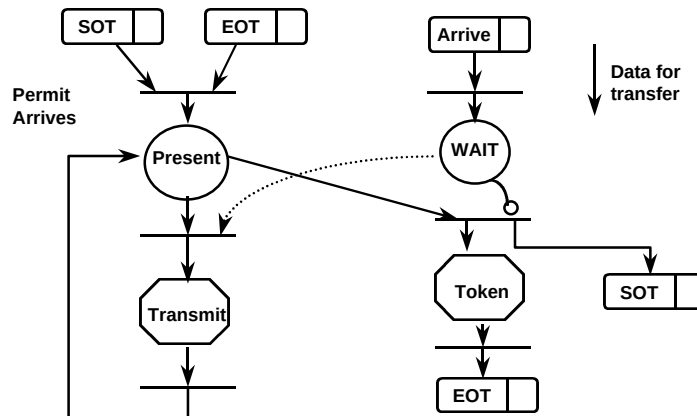
delay in μs

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

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

Detailed Example (Cont.)

Protocol 1 - Each node can transmit as much data as possible if it has a permit to do so

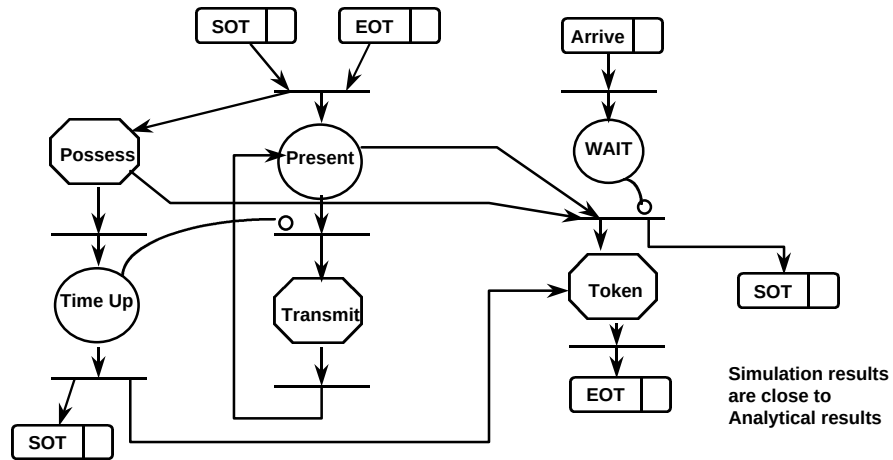


© IEEE 1988

[Jackman88] 69

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

Protocol 2 - A fair protocol with TIME-OUT



**Simulation results
are close to
Analytical results**

Copyright © 1995-1999 SCRA

© IEEE 1988

[Jackman88] 70

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.



“Network-level” or Reactive Design with Statemate



- | **Statemate - A graphical specification mechanism for reactive systems from i-Logix**
- | **Characteristics of a reactive system**
 - m **Continuously interacts with its environment - event-driven**
 - m **Able to respond to interrupts**
 - m **Real time constraints**
 - m **Provides high level of concurrency in system operation**

Copyright © 1995-1999 SCRA

71

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



- | **Usually time-driven and export parameters to the network level**
- | **Same “structure” as the network-level environments**
- | **Consist of model libraries, block diagram editors, simulation managers, postprocessors, database manager, and consistency checker**
- | **Model libraries contain**
 - m **Signal sources**
 - m **Modulators/demodulators**
 - m **Encoders/decoders**
 - m **Channel models**
 - m **Arithmetic and logic operations**

Compiled to a C/FORTRAN
Program for execution

Copyright © 1995-1999 SCRA

73

Link-level design environments.

The Execution Manager

- | **DSP algorithms are modeled by data flow graphs similar to Fully Specified Flow Graphs (FSFGs)**
- | **Data flow graphs: execution model**
 - m Fully dynamic DFGs
 - m Static allocated DFGs
 - m Self-timed DFGs
 - m Fully static DFGs
- | **More on this topic in the scheduling and assignment module**

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

The Execution Manager (Cont.)

I Two tasks undertaken by the Execution Manager

m Assignment: Assigning tasks to processors

m Scheduling:

q Precedence

q Start times of tasks

Increasing
Application-
specificity
& Domain
Knowledge

	Assignment	Precedence	Start-Timing
Fully Dynamic	Run-Time	Run-Time	Run-Time
Static Allocation	Compile	Run-Time	Run-Time
Self-Timed	Compile	Compile	Run-Time
Fully Static	Compile	Compile	Compile

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



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

Copyright © 1995-1999 SCRA 76

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.

Block Oriented Simulators

| Block-oriented simulators

- m Matlab/Simulink
- m Mathematica
- m Khoros
- m Ptolemy/Blossim
- m BOSS/SPW
- m COSSAP

Primarily BLOCK-ORIENTED simulators capable of
TIME and FREQUENCY domain operations

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



Khoros 2.0



- | **Khoros is a complete data exploration and software development environment**
- | **Advantages**
 - m Reduces time to solve complex problems
 - m Provides free sharing of code modules
 - m Promotes portability
- | **Primary emphasis**
 - m Software development
 - m Data visualization
- | **KHOROS software is divided into several toolboxes**
- | **Toolboxes are organized by function and common objective**

Copyright © 1995-1999 SCRA

78

Khoros is widely used in image processing applications.



Software Organization



- | **Software organization is based on toolbox and software objects**
 - m Craftsman: toolbox management tool
 - m Composer: software object editor
 - m Guise: GUI design tool
- | **Craftsman, Composer and Guise are stand-alone tools**
- | **Cantata is the visual programming language environment**
 - m Complete software development environment
 - m Data visualization tool
 - m Algorithm layout tool

Copyright © 1995-1999 SCRA

79

Additional information on Khoros may be obtained at
<http://www.khoros.com>



Flexibility



- | **Easily configurable to include desired toolboxes and programs**
 - m Simple to add new toolboxes and programs to system
- | **Use of code generators integrated into Composer**
 - m Composer writes interface code for new programs to reduce programming time
 - q I/O structure, library includes etc.
- | **Data processing “operations” based on multi-dimensional polymorphic data model**
 - m Same operator (i.e., FFT) works on 1D, 2D, 3D,... signals

Copyright © 1995-1999 SCRA

80

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



Flexibility (Cont.)



- | **KHOROS 2 programs operate on data independent of storage format**

- m **Example**

- q **Can add 256x300 color Sun raster image with 512x512 VIFF-formatted image without any conversion**

- | **KHOROS 2 can handle large data sets**

- m **Example**

- q **Can take FFT of 2K x 2K x 2K float volume**

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



KHOROS Availability



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

Copyright © 1995-1999 SCRA

82

Additional information can be obtained at <http://www.khoros.com>



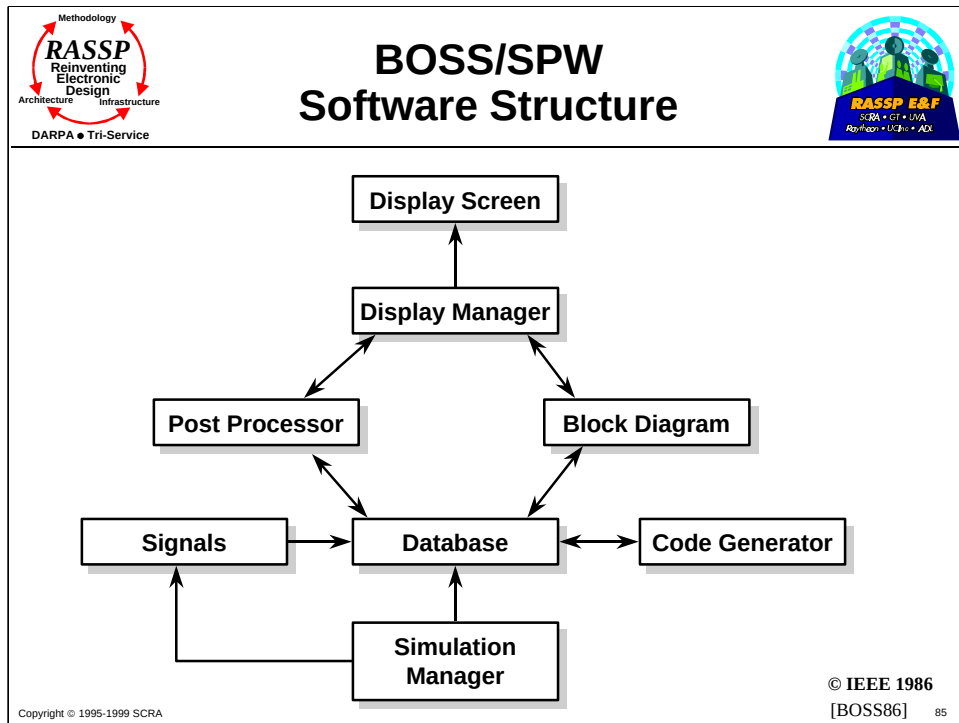
BLOSIM/Ptolemy



- | Precursor to Ptolemy (<http://ptolemy.eecs.berkeley.edu>)
- | User writes code for functional blocks, BLOSIM allows blocks to be interconnected and executed as ONE EXECUTABLE in C
- | Equivalent to writing a special-purpose simulation for an application, but standard interface allows blocks to be reusable. Allows multiple sampling rates
- | Ptolemy adds C++ and links to execution on parallel processors, adds model libraries, and adds a few additional domains
 - m Essentially a multi-level simulation model described earlier
 - m Attractive features - Matlab blocks can be included

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

[Messerschmitt84] 84



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



Module Outline

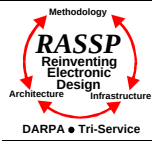


- | Introduction
- | Requirements Capture
- | Fixed-Point Design - A RASSP Approach
- | Simulation-Based Algorithm/Functional Design
- | Network-Level DSP
- | Link-Level DSP
- | Signal Processing Simulators
- | **PGM/PGSE**
- | RASSP Software Generation
- | Summary

Copyright © 1995-1999 SCRA

86

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).



Section Outline



| PGM/PGSE

- m PGM/PGSE overview
- m Application development in PGM
- m Issues in the use of PGM

We now introduce PGM and PGSE environments with examples.



Processing Graph Method, PGM



- | **The PGM allows to develop signal processing applications without any knowledge of the underlying machine architecture**
- | **PGM achieves this goal by providing**
 - m **A high level specification**
 - m **A graph oriented language**
 - m **Tools for translating the graphs into load modules for target machines**
 - m **And a run-time support environment which expands the graph instances**

Copyright © 1995-1999 SCRA

88

PGM facilitates an executable specification of the application.

PGM Fundamentals

- | **PGM is based on a modified “data flow” methodology**
- | **It differs from the classical data flow in the following**
 - m **The input queue threshold can be set more than 1**
 - m **An offset can be specified to skip a certain number of input data from the queue before reading data**
 - m **The number of elements to read into a node can be specified**
 - m **The number of elements to consume from input queues can be specified**
 - m **PGM is Data Flow only down to the level of scheduling nodes for execution**

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.



PGM and DSP



- | **PGM is to be used to build up signal processing applications**
- | **The signal processing application is defined as a set of graphs and command programs**
- | **The graphs are analogous to flow diagrams used to summarize signal processing flow**
- | **The command programs define the graphs interaction among themselves and the outside world**
- | **The result is a set of graphs and command programs that are translated into load modules which are subsequently executed under the PGM runtime environment**

Copyright © 1995-1999 SCRA

90

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.

The Graph

- | **A graph is a description of the signal processing algorithm for a particular function**
- | **The basic elements of a graph are**
 - m **Queues, representing the directed information flow through the graph**
 - m **Auxiliary data storage entities holding additional information**
 - m **And nodes representing the primitive signal processing elements of the graph**
- | **PGM supports a hierarchical form for the graph description**

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.

The Queues

- | **Queues provide the primary data storage and transfer medium for the graph**
- | **The queues are implemented as expandable FIFO**
- | **There are two kinds of queues in PGM**
 - m **Data queues, used for passing data between two nodes**
 - m **Trigger queue, used for passing channel synchronization signals**
- | **Trigger queues can force a synchronization of an otherwise asynchronous, data driven sequence of operations**

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.

The Auxiliary Data Storage

- | **The auxiliary data storage entities are used to hold additional information which would be inconvenient to send using queues**
- | **They are different from queues in that they hold a single datum of the declared mode at a time**
- | **PGM has two auxiliary data storage entities**
 - m **Graph Variables(GV)**
 - q **Internal graph variables, defined within a graph definition with local scope**
 - q **A dynamic graph variable, defined within a command program and its scope is all graphs and other command programs to which the command program passes its identifiers**
 - q **And Graph Instantiation Parameters(GIP), a start time constant passed at instantiation or or defined within the graph definition**

Additional data structures used in PGM are described.

The Node

- | **The node is the basic signal processing entity in PGM**
- | **It is executed when its scheduling criteria have been met**
- | **There are three scheduling criteria**
 - m All input queues must meet or exceed their thresholds
 - m Machine resources must be available
 - m Downstream queues must have enough space
- | **A Node is made up of a**
 - m Primitive
 - m Port(s)
 - m And a Primitive Interface Procedure (PIP)

Nodes store the computational primitives and their interfaces.



The Primitive



- | **The primitive is the signal processing entity within the node**
- | **It takes input values passed to it by PIP and calculates the output values which it passes back to PIP for appropriate action**
- | **EMSP Common Operational Support Software, ECOS, primitive specification contains information on each primitive that is available to the PGM programmer**

The primitive is the mathematical operation.

Ports

- | **The port is a logical concept defined as the point at which data enter or leave the node**
- | **The classification of ports depends on whether the data are**
 - m **Input or output from the node**
 - m **From or to queue or auxiliary data storage entity, or**
 - m **Used by the primitive alone or by the PIP**
- | **A node must have**
 - m **a port attached to an input queue to allow for a node scheduling criterion**
 - m **An output port from which more than one datum of the declared mode must be to a queue**

Ports are part of the node interface

Primitive Interface Procedure

- | **The Primitive Interface Procedure (PIP) provides the primitive with appropriate values from the node's various ports**
- | **The PIP allows for data transfer into and out of the primitive by providing**
 - m **A logical connection between the primitives input and output and the input and output queues**
 - m **Removal and generation of trigger queue pulses**
 - m **Introduction and transmission of graph variable to or from the primitive**
 - m **Transmission of constants defined by start time expressions to the primitive**
 - m **Calculation of variable Node Execution Parameters (NEPs)**

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

- | **Subgraphs allow hierarchical structures to be contained within graph definitions**
- | **A subgraph acts like a macro definition in that a subgraph is replaced during the graph instantiation is replaced by the nodes and queues of which the subgraph is composed**
- | **A subgraph may be used several times in the definition of a graph in a manner similar to the way a primitive may be used in several different nodes in a graph**

Subgraphs allow the construction of super-nodes (hierarchy)

Merge

- | **Merge allows for data dependent data processing**
- | **Merge has several input queues, one being a control queue**
- | **The control queue contains information that tells the merge which one of several input queues is to be read, and then places data to a single output queue**
- | **Control queues are mode integer, they have an implied**
 - m **Threshold**
 - m **Read**
 - m **Consume**
 - m **And offset, zero**

Description:

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

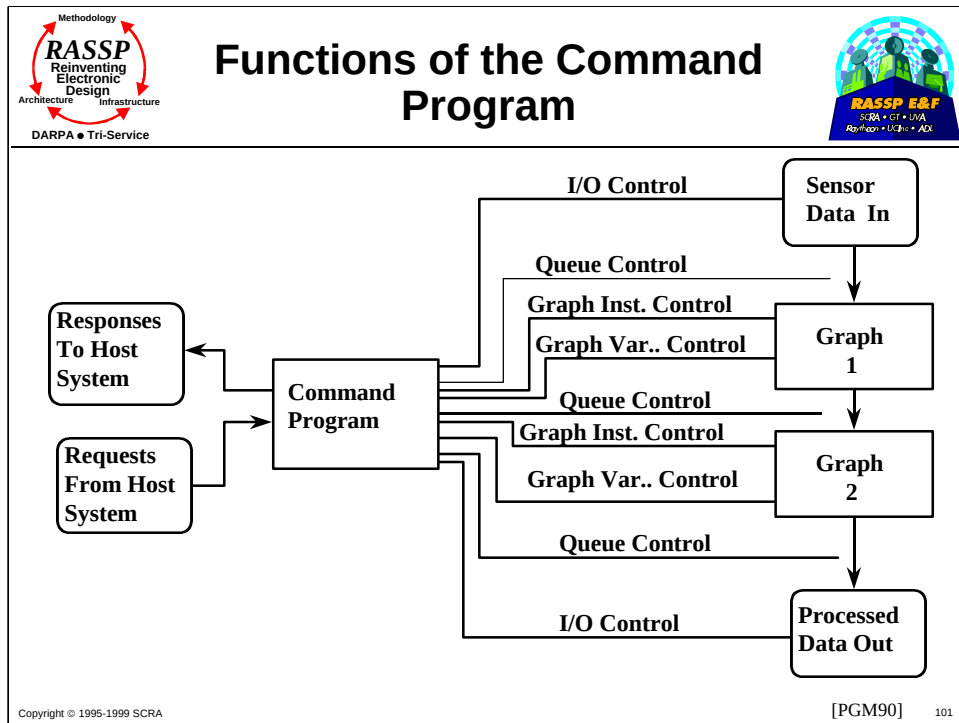


Command Program



- | **Command programs are application dependent and control graph execution and interaction**
- | **Command programs are to be used for control rather than signal processing**
- | **They are written by the application programmer, using a High Order Language (HOL) together with a set of procedure calls defined in the PGM specification**
- | **The HOL provides control structures within which the SPGN procedure calls are embedded**
- | **In addition, communication between the command program and the outside world is supported by the HOL and the underlying operating systems**

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.



Functions of the Command Program (Cont.)



- | **The command program has the capability of**
 - m **Defining a queue and its connections (Queue Control).** This capability is important because the graph is defined as having no external queues
 - m **Defining the connection of such a queue to an I/O procedure starting or stopping the procedure (I/O Control)**
 - m **Interacting with a graph (Graph Instance Control) by starting, stopping, or reinitializing the graph,**
 - m **Or by changing the values of graph variables passed to the graph (Graph Variable Control)**

Copyright © 1995-1999 SCRA

102

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>

I/O Procedures

- | **I/O procedures are implementation and application specific processes that provide the capability for**
 - m Passing signal input data between external devices and graphs and command programs
 - m Passing processed output data between graphs and command programs and external devices
- | **An input I/O procedure takes data from one or more external devices, processes it, and outputs it to one or more dynamic queues**
- | **An output I/O procedure takes data from one or more dynamic queues, processes and outputs it to one or more external devices**

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>



PGM Runtime Environment



- | **The command program accomplishes its functions by making calls to the PGM runtime environment**
- | **In addition to providing system calls for the command program, the runtime environment takes care of**
 - m **Scheduling the signal processing tasks for execution**
 - m **Managing and allocating processing resources**
 - m **Performing memory management services**

Copyright © 1995-1999 SCRA

104

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>



PGM Overview Summary



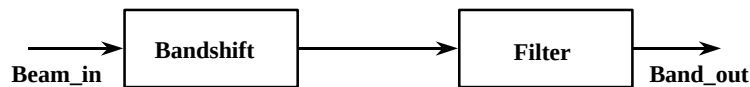
- | **PGM is based on a modified data flow methodology**
- | **A PGM application consists of graphs and command programs**
- | **PGM graph is composed of**
 - m Node (s)
 - m Queue (s)
 - m Sub-graphs
- | **PGM command programs perform**
 - m Queue Control
 - m I/O control
 - m Graph Instance Control
 - m Graph Variable Control

Copyright © 1995-1999 SCRA

105

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>

- | 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>



Application Development in PGM (Cont.)



- | **Each of the distinct operations in the previous block diagram corresponds to a node of a the PGM graph**
- | **The data passed between these nodes corresponds to the data queue**
- | **To change the block diagram to a PGM graph, the following steps are required**
 - m **Lay out the nodes and queues**
 - m **Select a descriptive name for each node**
 - m **Select a name for each queue**
 - m **Find the appropriate primitives to perform the required signal processing functions in the block diagram using the ECOS primitive specification**

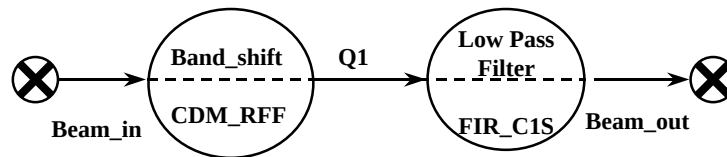
Copyright © 1995-1999 SCRA

107

Descriptions:

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

- | For the simple example given, there is a one-to-one correspondence between the block diagram and the ECOS primitives
- | The band-shift operation can be performed by
 - m CDM_RFF, a complex demodulation primitive
 - m Low pass filtering can be performed by the FIR_C1S, the FIR filter with decimation primitive



Description:

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



Application Development in PGM (Cont.)



- | **Each primitive in the ECOS has a data sheet that specifies**
 - m **Description**
 - q **Specifies the functionality of the primitive**
 - q **The method of implementation**
 - q **Errors inherent in the algorithm**
 - q **Reference for the algorithm**
 - m **Pseudo Code**
 - m **Parameter List, a listing of all input and output parameters with descriptions**
 - m **A list of constraints for the primitive**
 - m **Performance Features**
 - m **Performance Guidelines for Exception Handling**
 - m **Execution time formulas**

Copyright © 1995-1999 SCRA

109

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>

Application Development in PGM (Cont.)

- Once the appropriate primitives have been found, the next step is to define all graph entities involved in the storage and passing of data, this is done by means of Queue and Variable Attribute Table (QVAT)

Entity Name	TYPE GIP VAR. QUEUE	SCOPE		MODE	Initial Values	Description
		LOCAL FORMAL	IN			
			OUT			
A1	GIP	Local		FLOAT Array(7)	-1.0E-1, 1.0E-1 2.5E-1, etc....	Filter Weights for the FIR filter

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.

Application Development in PGM (Cont.)

- | The next step in the process is to define all the connections among the different nodes of the graph: this is done by means of node attribute table
- | For each node in a graph, there should be a corresponding node attribute table

Node Name	Primitive Name	INDEXING		Description	
FIR_NOD	FIR_C1S	Read	V Offset	V Consume	Finite Impulse Response Filter
PIP_INs	Threshold				Description
PRIM_INs	Threshold	V Read	V Offset	V Consume	Description
Q1	137	137	0	128	Input Data
PRIM_OUTs				V Valve	Description Output Data
PIP_OUTs				V Pulse/Produce	Description

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

Copyright © 1995-1999 SCRA

112

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>



Application Development in PGM (Cont.)



I Command program control

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

Copyright © 1995-1999 SCRA

113

For additional information please consult
<http://pms428.uswinfo.com/pgm-1.htm>



Application Development in PGM (Cont.)



- | **Input/Output control provides three procedures for interacting with graph I/O**
 - m **%INITIO**, provides for connection of an I/O channel with one or more I/O queues
 - q Return a **IO_Procedure_id** used subsequently to identify the procedure for starting and stopping the I/O
 - m **%STARTIO**, starts a previously defined and initialized I/O
 - m **%STOPIO**, suspends a previously started I/O
 - q The I/O process can be started again with a **STARTIO** procedure

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**
 - m **Create a particular graph instance with particular**
 - q **Queues**
 - q **Nodes**
 - q **Graph variables**
- | **It is possible to have many different graph instances that started from the same graph realization active at the same time**

Copyright © 1995-1999 SCRA

115

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



Application Development in PGM (Cont.)



- | **The following command program, SPGN, procedures are available for graph instance control**
 - m **%START**, returns a Graph_id that is used by the command program to refer to a particular instance of the graph
 - m **%STOP**, stops a graph instance
 - q **The stopping of the graph instance terminates and destroys a particular graph instance**
 - q **Disconnects, but does not destroy, any queue from the graph**

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



Application Development in PGM (Cont.)



- | **There are eleven procedures available under command program, SPGN, for manipulating queues**
 - m **%CreateQ**, causes a queue to be created dynamically
 - m **%DestroyQ**, destroys a dynamically created queue
 - m **%INITQ**, Initializes a dynamically created queue
 - m **%FLUSHQ**, removes all data elements from a queue
 - m **%CONNECTQ**, connects the command program to head or tail of a previously created queue
 - m **%DISCONNECTQ**, releases the queue from the command program
 - m **%ADDDATA**, adds data to a queue behind the data already in the queue
 - m **%READQ**, reads a number of data elements from a queue to a command program, when enough data present.
 - m **%WAIT**, waits for a specified event to occur
 - m **%UNLINK**, disconnect dynamically created queue from a graph
 - m **%LINK**, allows dynamically created queues to connect to graph I/O ports

Copyright © 1995-1999 SCRA

117

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**
 - m **%CREATEGV**
 - q Dynamically creates a graph variable and returns an identifier which it can be referenced
 - m **%DESTROYGV**
 - q Destroys a a previously dynamically created graph variable
 - m **%READGV**
 - q Enables the command program to read a data element or group of array elements from a graph variable
 - m **%WRITEGV**
 - q 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**
 - m Family structure to specify channelization
 - m Fixed and variable parameters to govern the processing within each node of PGM graphs and relative node execution rates, and sequences of the nodes
 - m 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.**

Copyright © 1995-1999 SCRA

119

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

[MMC/LMC 94]



Issues in the use of PGM



- | **There are some processes that do not lend themselves to data flow specification. How will PGM accommodate these?**
- | **The overhead associated with scheduling and data flow, communication, may be intolerable for some high data rate applications?**
- | **The only available run time support for PGM at this time is the AN/UYS-2 which provides hardware support that is not portable to RASSP Model Year Architectures (MYA). The development cost of such run time systems would rise the application development cost considerably**

Copyright © 1995-1999 SCRA

120

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

[MMC/LMC 94]



Issues in the use of PGM (Cont.)



- | **The data flow laws for deterministic implementation of discrete processes on a continuous data stream are well established**
- | **The data flow laws specify scheduling data dependencies, schedule sequences, and data transfer**
- | **Continuous data streams of sensor signals are exactly the class of signals that systems developed with RASSP tools are designed to process**
- | **In addition, PGM provides mechanisms for the odd auxiliary computation that may be necessary within a signal flow application**

Copyright © 1995-1999 SCRA

121

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**
 - m Sequential lists of primitive executions that execute a cycle of the translated graph
 - m Time line programs
 - m Parallel forms of list or time line programs
- | **Applications may be translated to**
 - m Completely static form
 - m 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

Copyright © 1995-1999 SCRA

122

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

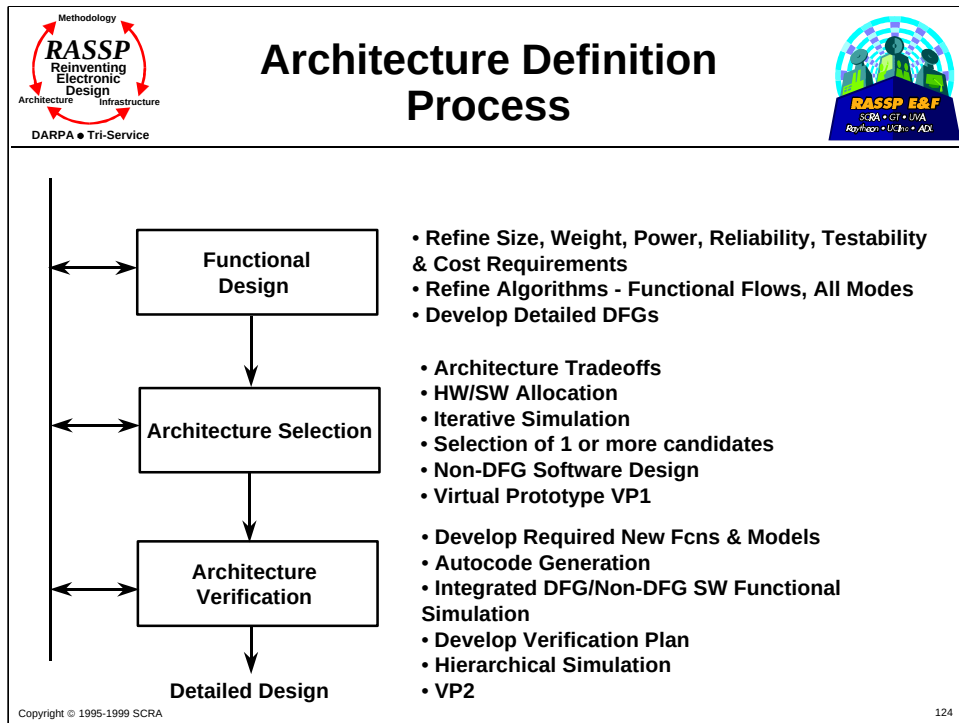
[MMC/LMC 94]



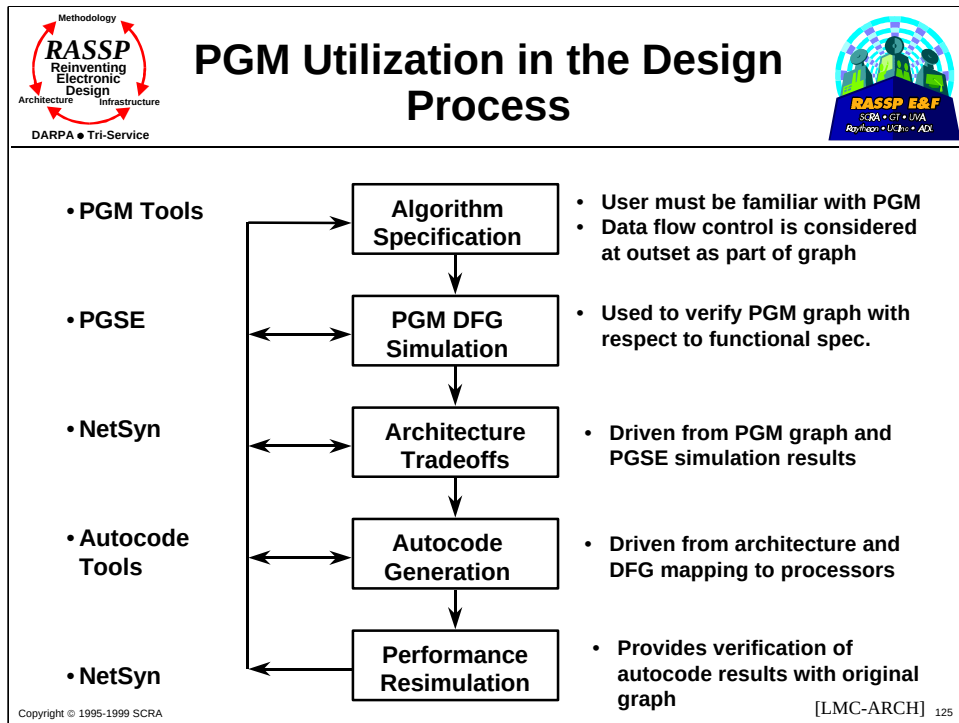
Module Outline



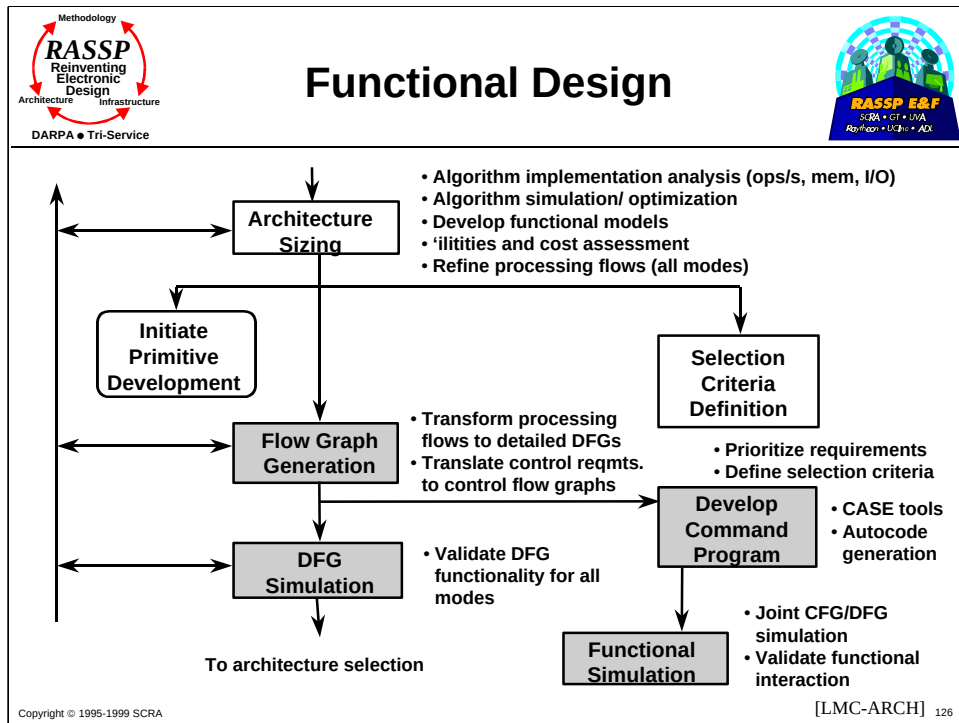
- | Introduction
- | Requirements Capture
- | Fixed-Point Design - A RASSP Approach
- | Simulation-Based Algorithm/Functional Design
- | Network-Level DSP
- | Link-Level DSP
- | Signal Processing Simulators
- | PGM/PGSE
- | **RASSP Software Generation**
- | Summary



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 Process in RASSP



- | **Algorithm flows specified in SPW, or any other higher level math tool, form are captured as PGM graphs**
- | **The initial graph is functional only, expressing the mathematical operations in the algorithm flow as nodes or subgraphs and the data flow as queues**
- | **Recording queue contents during execution will provide intermediate result for comparison from those of the SPW representation**
- | **This functional graph is then expanded into a full application specification**

Copyright © 1995-1999 SCRA127

Algorithm design refers to functional design.

[MMC/LMC 94]



Application Development Process in RASSP

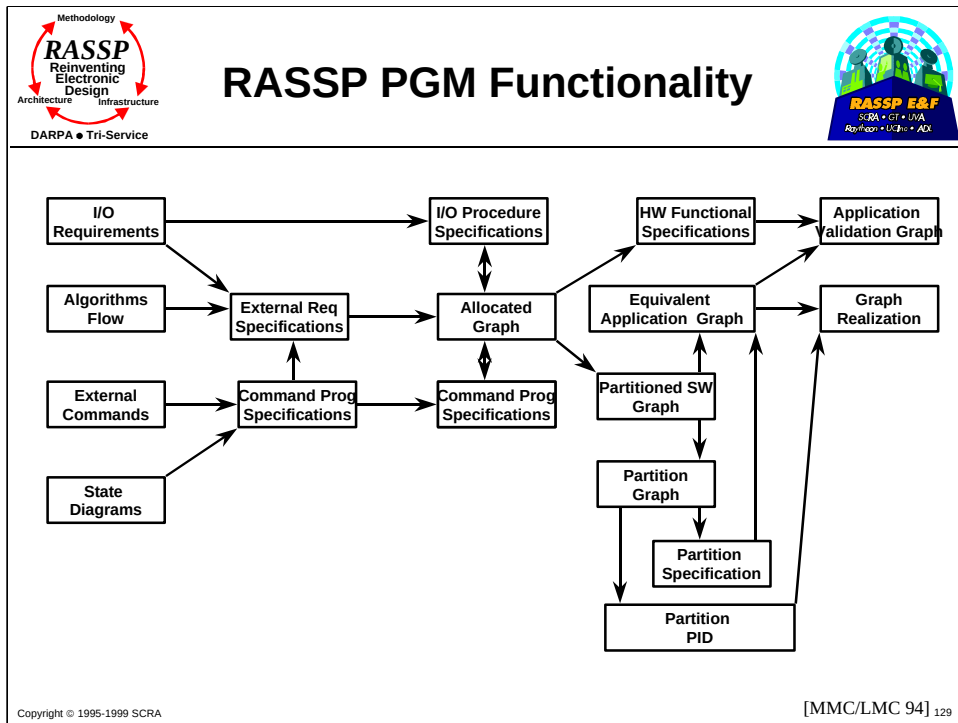


- | **Topology is expanded to represent**
 - m All possible data paths
 - m Controls are specified for all operating modes
 - m Sets or ranges of values for all variables are specified
- | **Graphical specification and execution tools are used to produce a full PGM specification of the applications**
- | **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**

Copyright © 1995-1999 SCRA

128

[MMC/LMC 94]



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**
 - m Are block diagrams depicting the high-level computations associated with an application, e.g. SPW, MATLAB
- | **I/O requirements**
 - m Identify the amount of throughput the application will have to handle, modes of data, and the source(s) of the data
- | **External commands**
 - m Are formal definitions of all commands allowable to the operator
- | **State transition diagrams**
 - m 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

Copyright © 1995-1999 SCRA

130

[MMC/LMC 94]



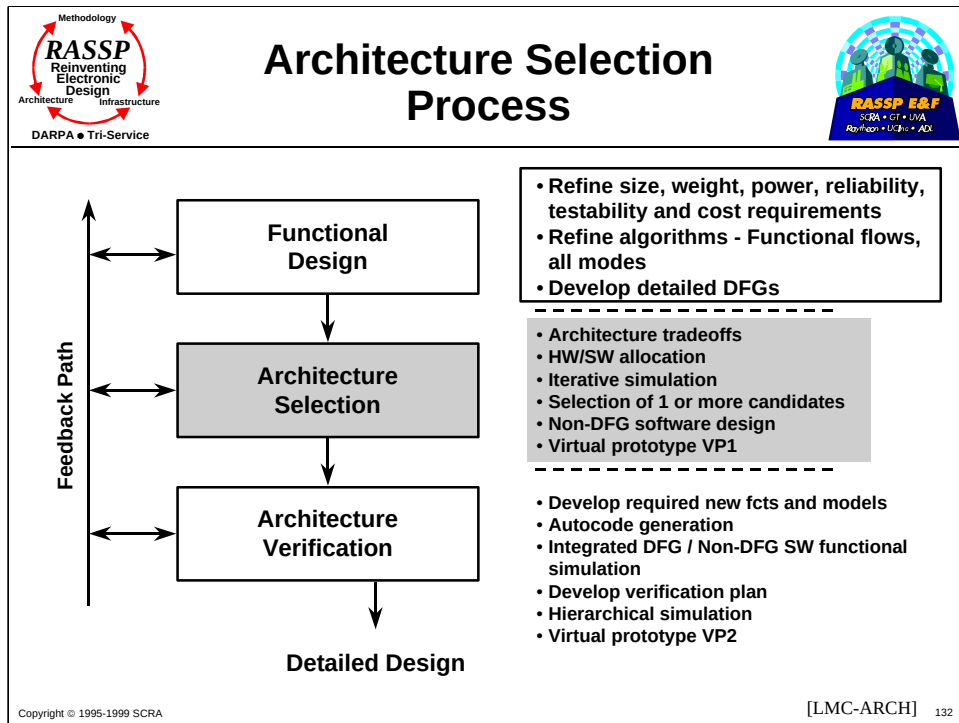
Application Development Process in RASSP: Requirement Refinement



| Executable Requirements Specification

- m 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
- m Command Program Specifications
- m 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.

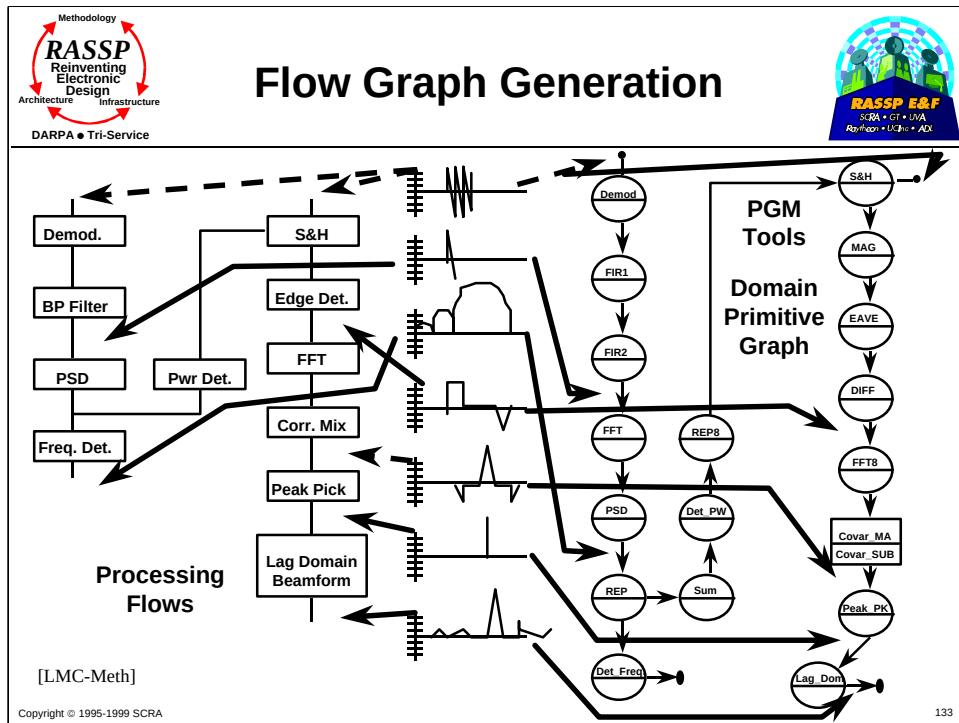
The architecture definition process is a new HW/SW codesign process in the RASSP methodology for high-level virtual prototyping and simulation. The primary concern in the architectural definition process is to select and verify an architecture for the signal processor that satisfies the requirements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

The overall task is to:

- 1 Define and evaluate various architectures
- 1 Select one or more for detailed evaluation that appear to meet the requirements
- 1 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

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

[MMC/LMC 94]



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



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

Copyright © 1995-1999 SCRA

135

[MMC/LMC 94]



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



| Partition Graph

- m Is a stand-alone SPGN graph which represents either a hardware or a software partition

| Corepresentation Validation Graph

- m Is a PGM graph in which all partitions identified in the partitioned graph have been replaced with equivalent nodes
- m This graph is used to simulate the functionality and performance of the entire application graph on selected candidate architecture
- m Partition Specification
- m For each partition graph, a partition specification is produced providing an interface to the partition PID which is produced to represent the partition graph

[MMC/LMC 94]



Application Development Process in RASSP: Target Software Generation



| Equivalent Application Graph

- m 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
- m Graph Realization
- m Is the compiled version of the equivalent application graph

| Partition PID

- m A partition PID is the source code that represents a partition graph, written in the computational element's native language

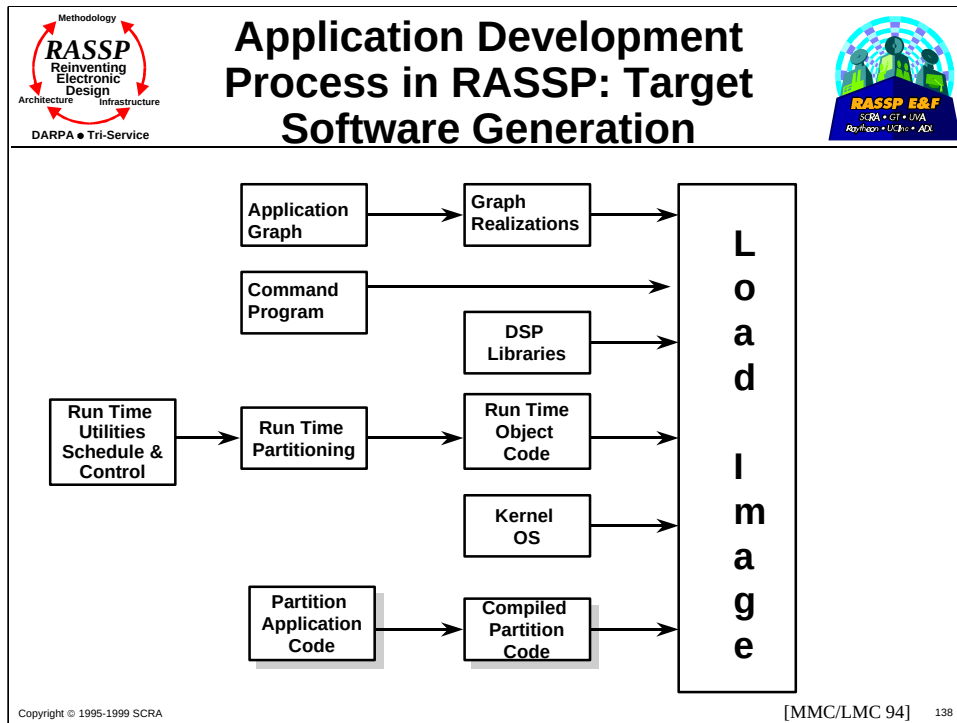
| Load Image

- m Is the end product of software development process

Copyright © 1995-1999 SCRA

137

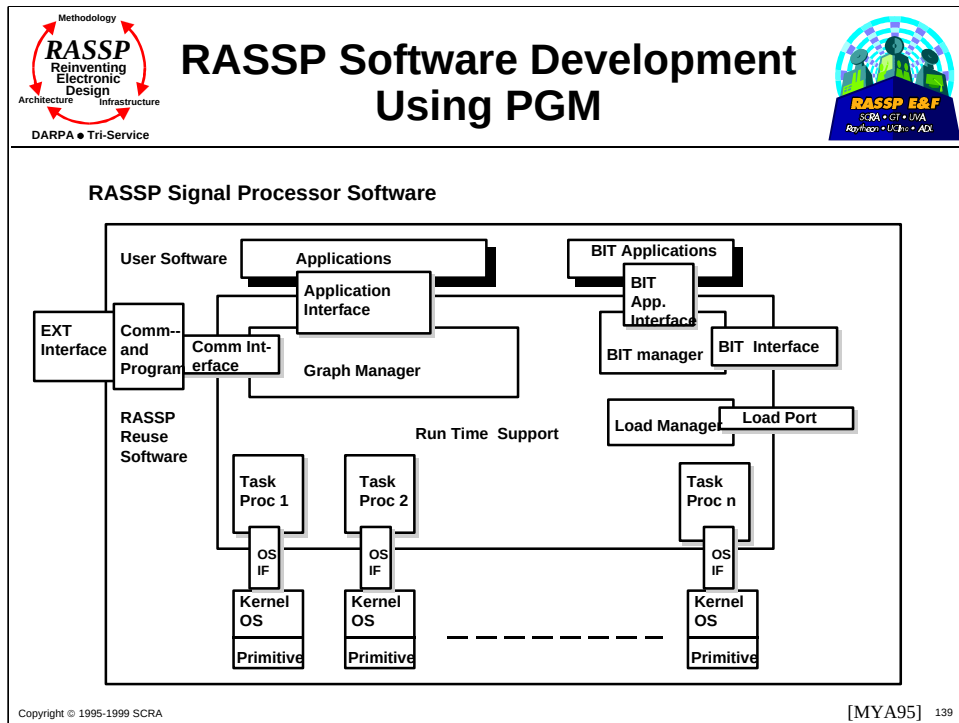
[MMC/LMC 94]



Description:

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

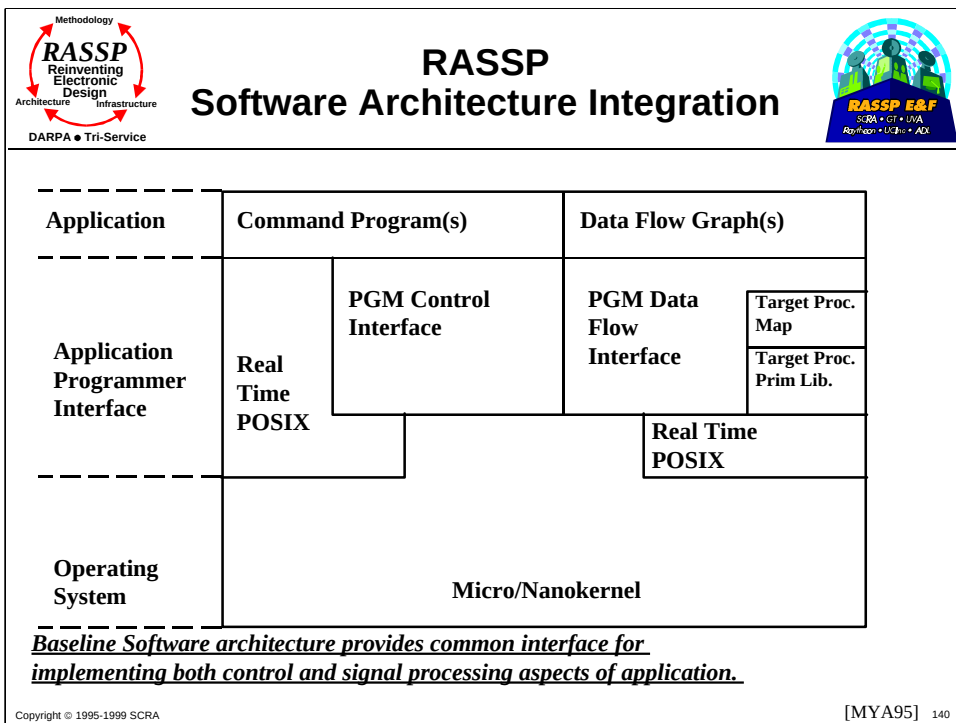
Top level application graph SPGN specification is compiled using the graph realization. The DSP libraries provide object code for specific implementation of some of the primitives. All run-time utilities, schedule and control routines are also partitioned and then presented to the load image box to be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



Tools for Architecture Selection and Verification



| **NetSyn: (Assignment/Performance Simulation)**

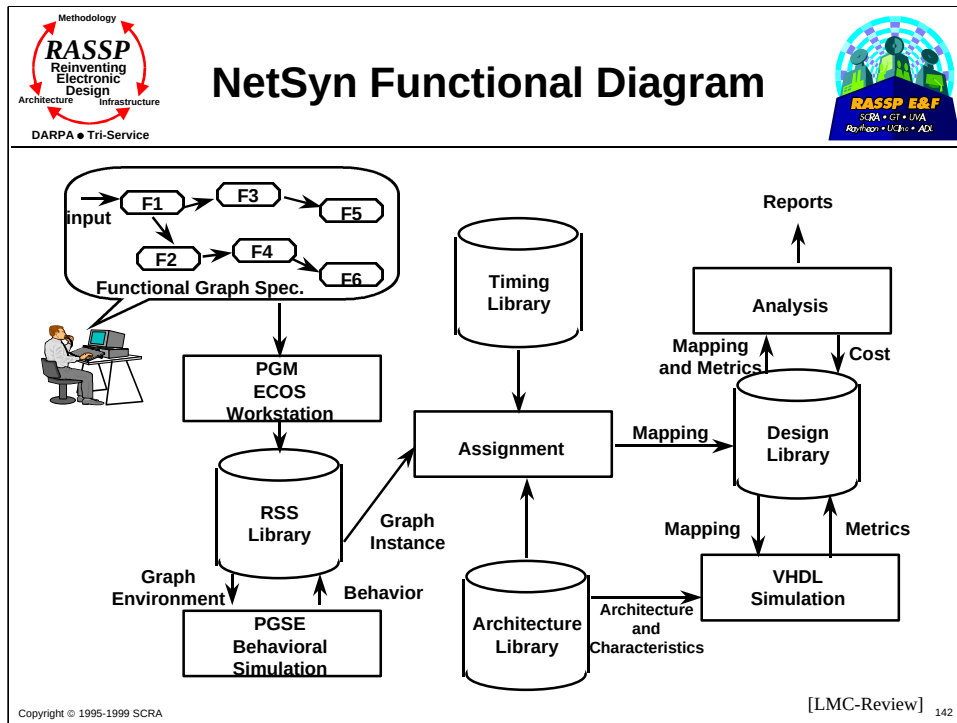
- m **Driven from PGM data flow graph and PGSE output**
- m **Performance simulation for candidate architectures**
- m **Library primitives**
 - q **PGSE executables**
 - q **Target dependent timing database**
- m **Outputs include reports, architecture, mapping, and graph**

Copyright © 1995-1999 SCRA

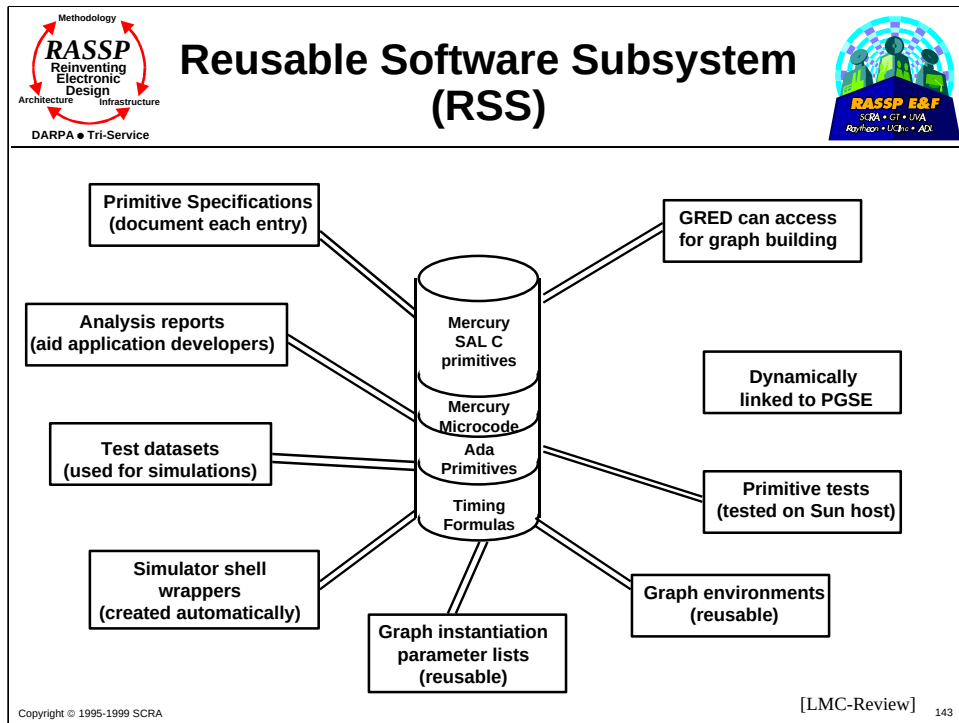
[MYA95] 141

[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.



Methodology
RASSP
Reinventing
Electronic
Design
Infrastructure
DARPA • Tri-Service

Reusable Architectural Parts Library



RASSP E&F
SCRA • GT • USA
Replicon • U.S. • ADX

- | Hierarchies of architectures are handled
- | Connection rules generate architectures from entities
- | Size, weight, power, cost values generated for architectures
- | Includes environment for testing behavioral models of entities
- | Timing functions included
- | Mercury entities and architectures included:
RACE

Copyright © 1995-1999 SCRA [LMC-Review] 144

[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.



Tools for Architecture Selection and Verification (Cont.)



| Autocode: (Code generation and run-time control)

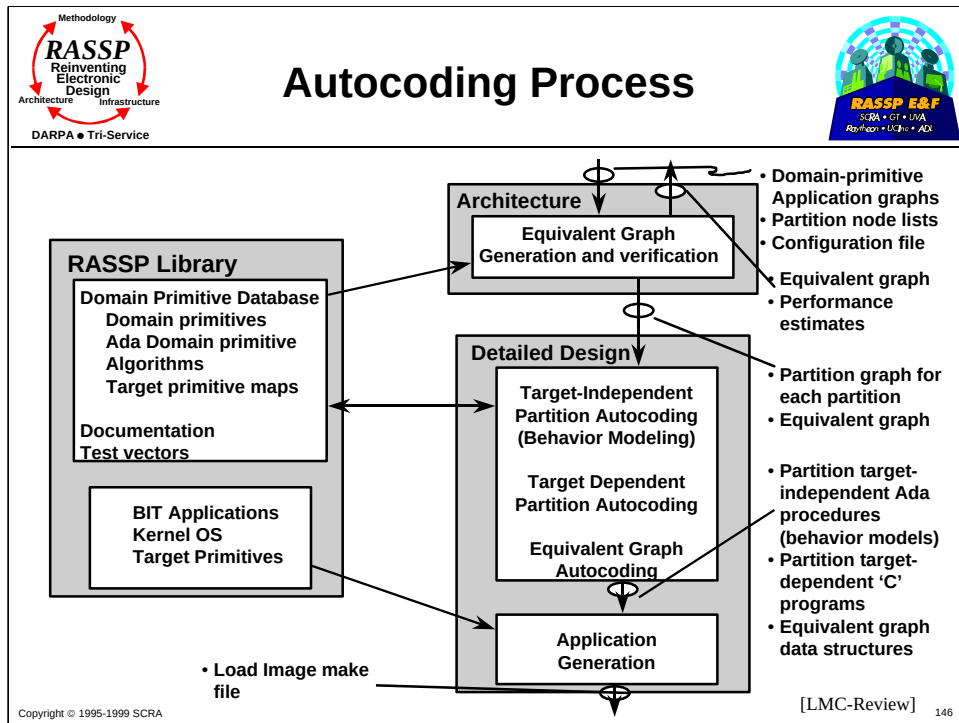
- m Driven from PGM data flow graph, architecture, and mapping
- m Library primitives
 - q Target independent PGSE executables
 - q Target primitive maps (TPMs) to specific target processors
- m Outputs include
 - q Modified graph
 - q Timing estimates for graph
 - q Target code compatible with run time system

Copyright © 1995-1999 SCRA

[LMC-Review] 145

[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.



Autocoding Process Inputs and Outputs



I Inputs to the autocoding process

- m PGM application graph SPGN file
- m Candidate architecture configuration file
- m Lists of nodes in processor software partitions
- m Lists of nodes in hardware partitions

I Autocoding outputs

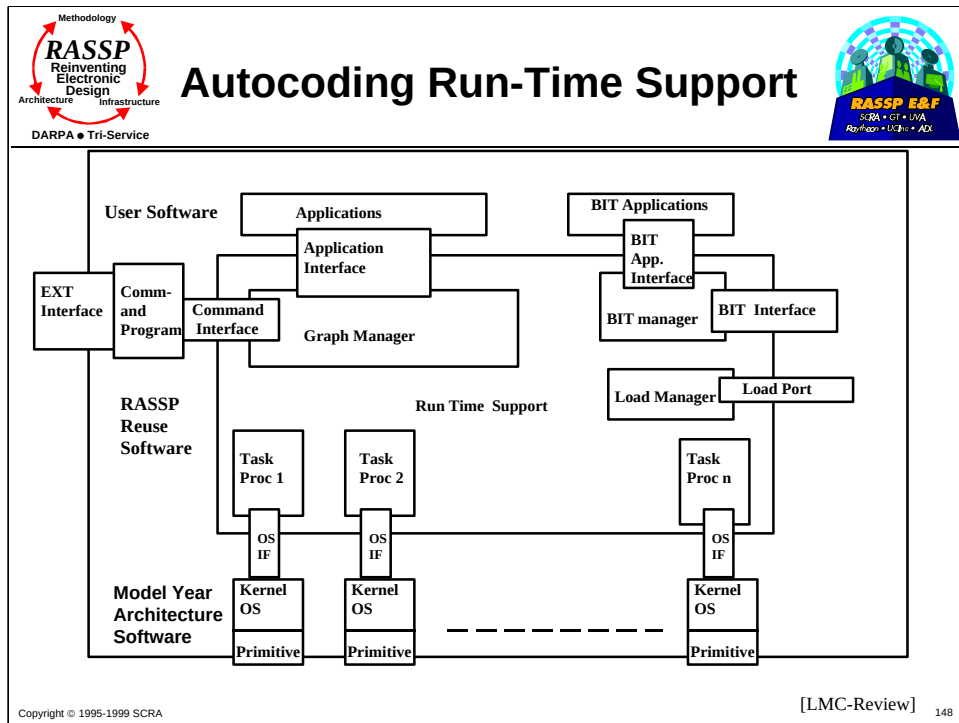
- m Partition "C" programs for target processors
- m Partition and application performance estimates generated
- m Run-time system load image for candidate architecture
- m Behavior models for hardware or software partition

Copyright © 1995-1999 SCRA

[LMC-Review] 147

[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.



Flow Graph Generation and Simulation Mechanism



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

Copyright © 1995-1999 SCRA

149

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.

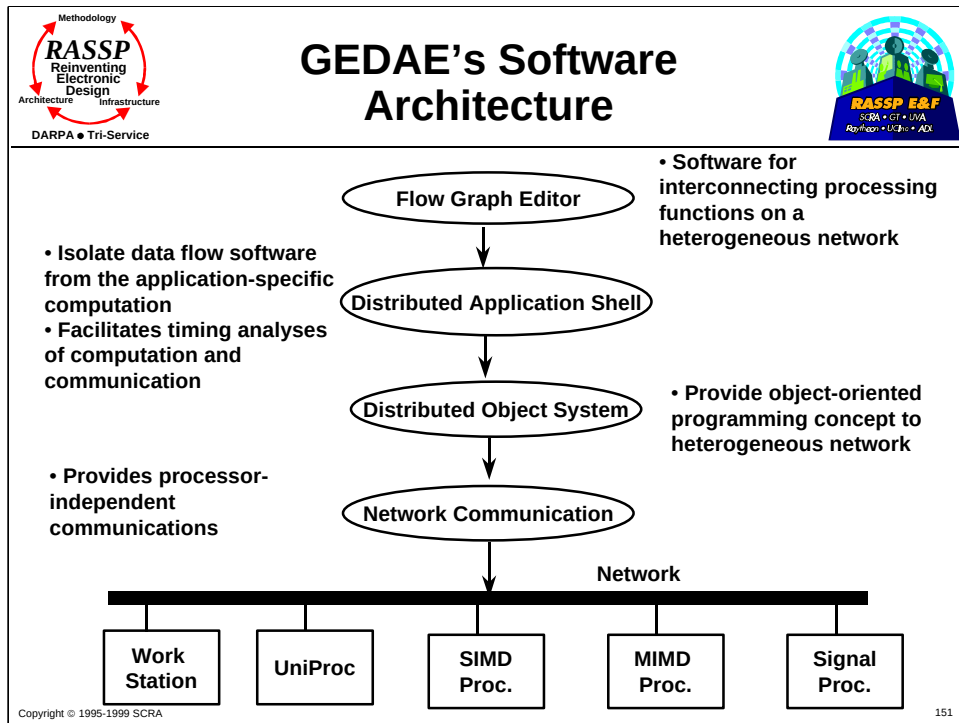


- | **The Graphical Entry Distributed Application Environment closely relates to PGM and its supporting tools.**
- | **The environment is used to**
 - m **Graphically construct application graphs**
 - m **Distribute the application graph on remote processors**

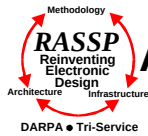
Copyright © 1995-1999 SCRA

150

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.

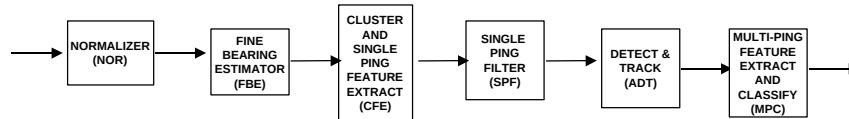


A Case Study of RASSP Software Development Using GEDAE



- **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

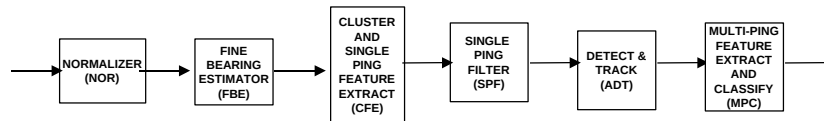
ETC Algorithm Description



I Front-end Processing (Acoustic signal processing)

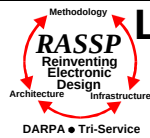
- m **Normalizer (Norm)**- compensation for underlying noise background, dynamic intensity variations such as may occur in reverberation-limited shallow water.
 - q The Coded Pulse (CP) normalizer is a single dimension normalizer, range only
 - q The Coherent Waveform (CW) normalizer is a two dimensional normalizer, range and Doppler
- m **Fine Bearing Estimator (FBE)**- high resolution bearing information
 - q Interpolation between beams with Doppler compensation
- m **Clustering and single ping Feature Extraction (CFE)**
 - q 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)

- m Single Ping Cluster Filtering (SPF)- weed out some of the echo returns from clutter
- m Automatic Detection and Tracking (ADT)- automates the detection of potential targets over multiple pings.
- m 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**
 - m Messaging backplane for data and parameter distribution
 - m Large use of global variables
 - m Non-segmented processing on a per ping basis
 - m Combination of data and control flow
 - m ~50,000 lines of code
- | **Migration**
 - m Analyzed c-code hierarchy (in-house tools)
 - m Extracted core processing code
 - m Encapsulated code into GEDAE at highest level
 - m Decomposed only high load DSP functions for distributing
 - m Parallelize and segmentize signal processing code
 - m 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**
 - m Supported migration while on development host

Copyright © 1995-1999 SCRA

156



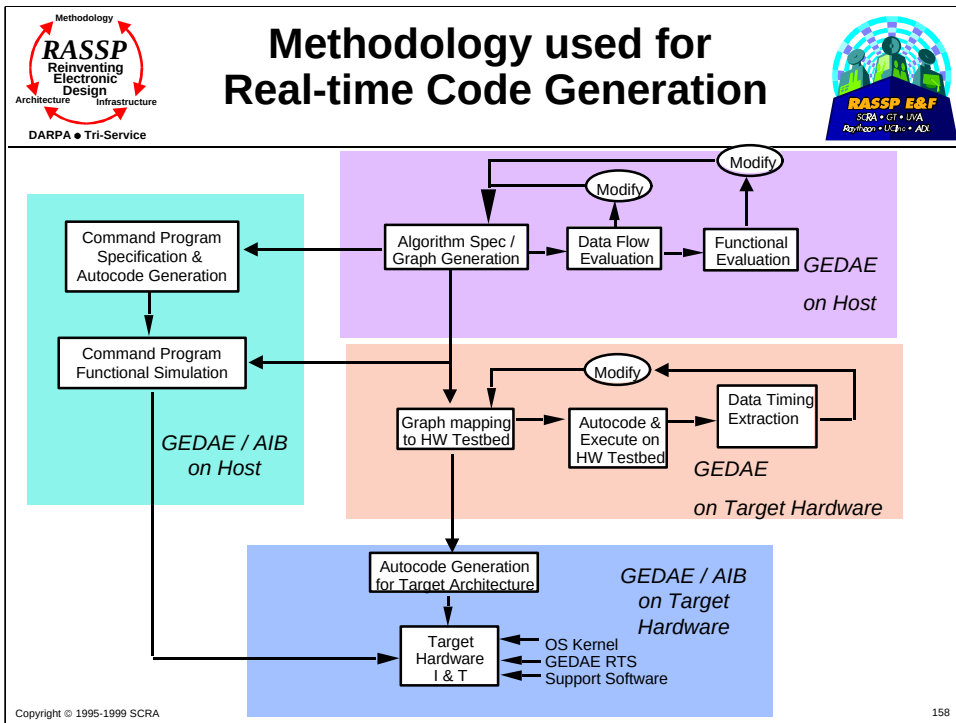
What This Team Considered As Critical Contributors to Success



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

Copyright © 1995-1999 SCRA

157





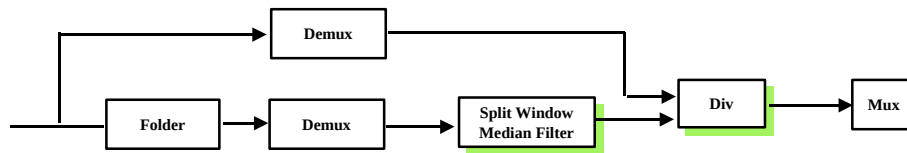
Graph Development Process



Graph Generation

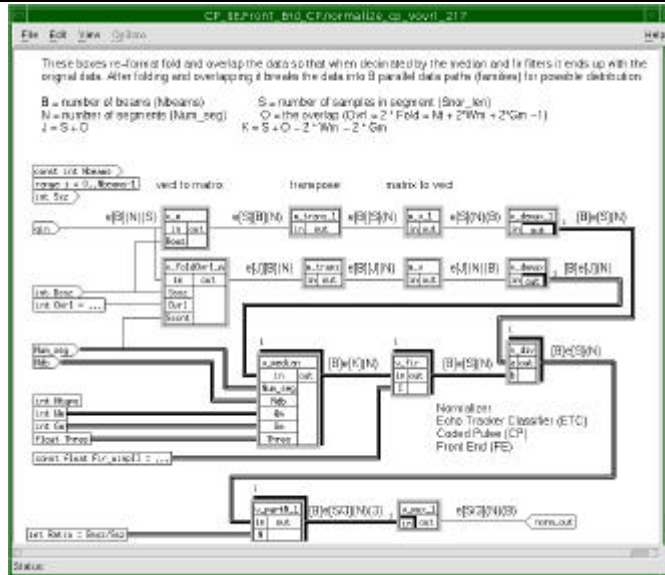
- **Determine Data Flow**
 - Scalar stream, vectors, matrices, variable vectors/matrices, structures
 - Instantiation
 - Queue sizing
- **Determine Functional Flow**
 - Custom vs. built-in primitives
 - Tradeoffs include - entry time, readability, maintainability, debug and test, runtime performance, 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 parallelize numerically intensive processing
- | Use V-arrays to support parameter dependent variations in data flow

Norm CP GEDAE Graph



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

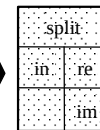
```
split (in, re, im) {
    derive output arrays
    ("re" and "im") according
    to some algorithms applied
    to input array ("in").
}
```

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



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

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

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);
}
```



Test and Integration Process



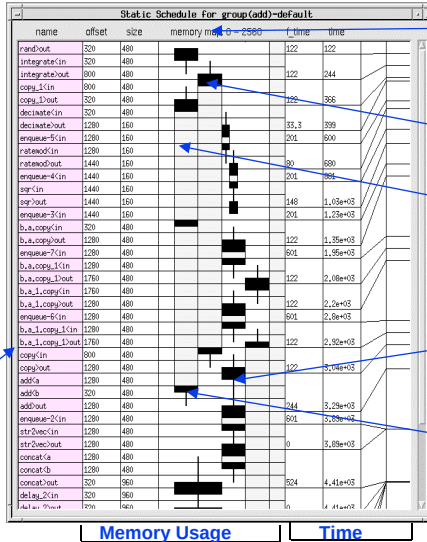
- | **Test Data Flow on Host**
 - m Primitive connection ensures data flow compatibility prior to test
 - m Box table/data table, show Q's and boxes graphically facilitate debug of data flow
 - m Optimization of memory
- | **Test Functional Flow on Host**
 - m Scope and/or files facilitates debug of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

Execution Sequence



Data generation

Data in use

Data storage

In-place calculation

Data consumption

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

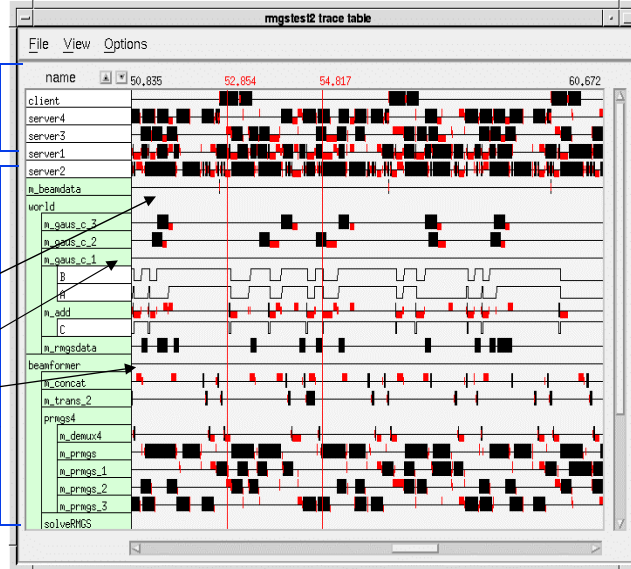
Hardware profile

Compute time

Data flow profile

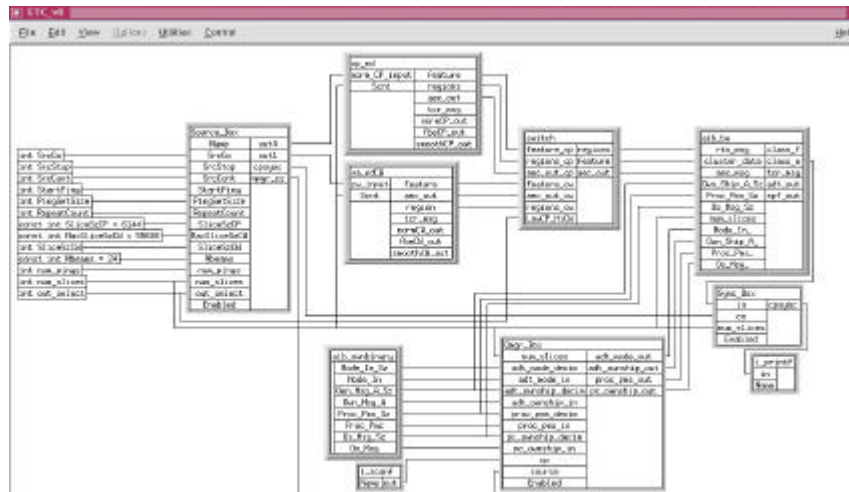
I/O time

Software profile



Copyright © 1995-1999 SCRA

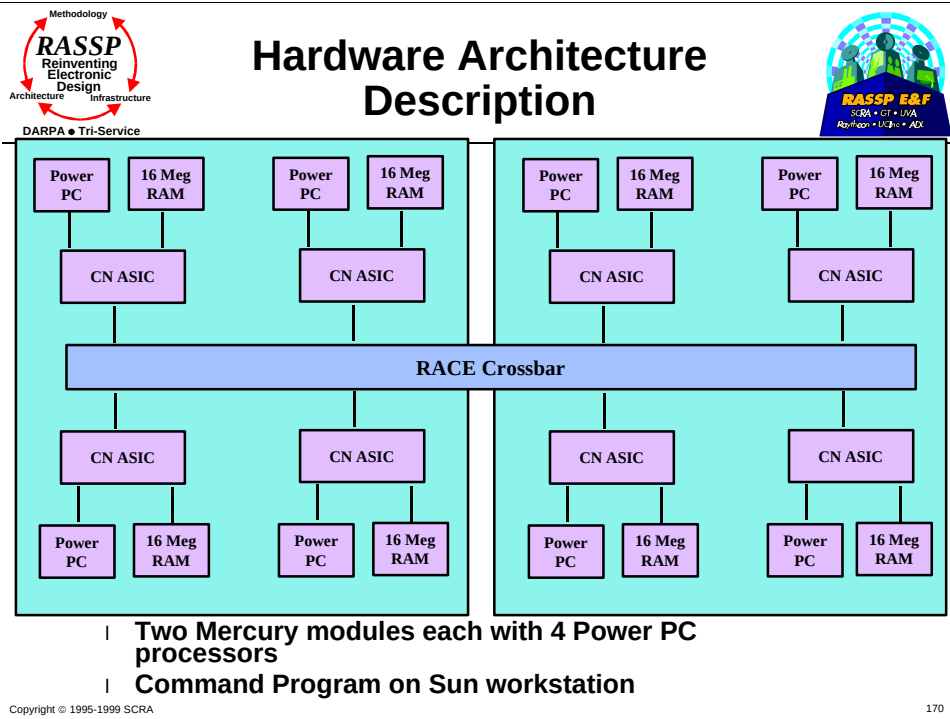
Top Level Graph



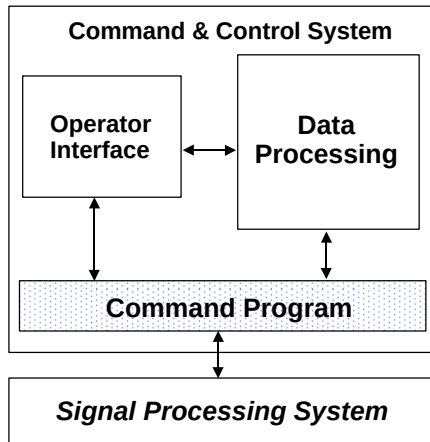
ETC4ALFS Top Level Graph

The graph is composed of

- | **Source_Box** which simulates real-time input data
- | **The Front end CP (cp_nd)** composed of
 - m **Norm,**
 - m **FBE**
 - m **CFE.**
- | **The Front end CW (va_ndCW)** composed
 - m **Norm**
 - m **FBE**
 - m **CFE.**
- | **A switch** to control flow of data to backend graph
- | **The Back end** composed of
 - m **SPF,**
 - m **ADT**
 - m **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



Command Program



Fundamental Command Program Requirement

- m 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)
- m Forward signal processor results

AIB - Application Interface Builder

Builds Application Programmer Interface (API)

- m Each mode is an object
- m Methods for mode entry/exit
- m Methods for submode entry/exit
- m Methods for execution state transition
- m Methods for buffering operator/DSP data

Input

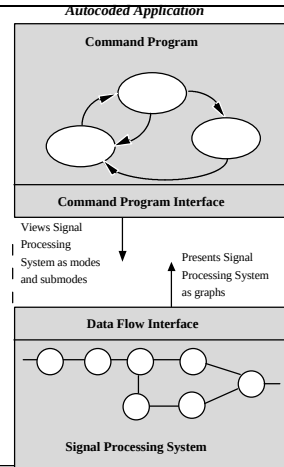
- m Enhanced top level graph declarations
- m 90% previously generated by DSP autocode process

Output

- m C source

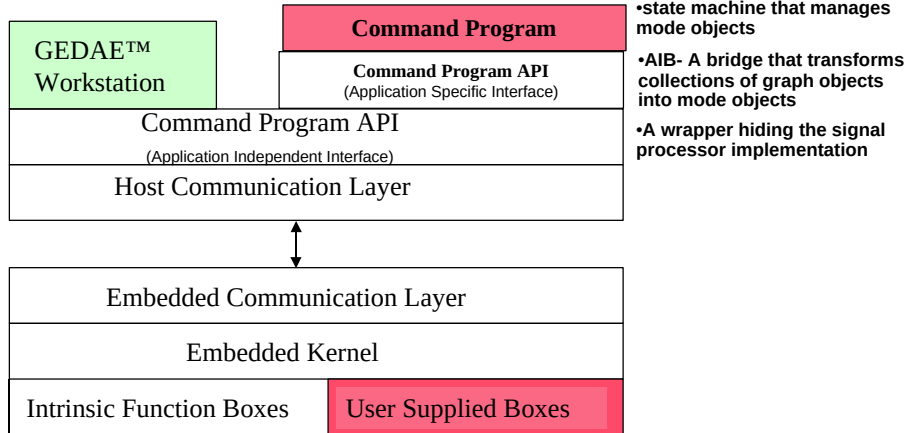
Future

- m Integrate with GEDAE GUI
- m Tighter integration with ObjectGeode



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

Layer and Conquer





ETC GEDAE Conversion Summary



- | **Training - 1 week**
- | **CP frontend captured, tested, and optimized on Host - 3 weeks**
 - m CP Front End is well partitioned into primitives
- | **CW frontend captured, tested, and and optimized on Host - 5 week**
 - m CW Front End is well partitioned into primitives
- | **Back End captured, tested, and optimized on Host - 2 weeks**
 - m 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.**
 - m tools were easy to learn and use
- | **Concerns of team with “black box” effect of autocoding software turn out to be unfounded**
 - m visualization aids were very effective
 - m insight to what the software was doing was actually better than 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 than a year.**

Tool Improvements

- | **Tool improvements made during project development**
 - m **Command program interface (AIB) added**
 - m **Improvements to facilitate data flow and test**
 - q **Automatic resizing of queues (Q) in GEDAE to minimize memory requirements and help isolate data flow problems**
 - m **Improved and added primitives for V-array support especially for dynamic (variable consume and/or produce) and non-deterministic (variable threshold) boxes**
 - m **System build (make) process improved**
- | **Future improvements or additional features we would like to see**
 - m **Better documentation of tool needed for embedding process, documentation for graph development on host is good**
 - m **File management system is limited for large multi-user program development**
 - m **More refined methods of shared data management to reduce amount of memory usage**
 - m **Final documentation generator of graph, parameters, and command program interface would be very useful and save time**
 - m **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**
 - m Executable specifications
 - m High-level test benches
 - m Simulation-based design
 - m Graphical and block-oriented algorithm capture
 - m 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.
- [Madiseti94] Madiseti 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., Gadiant, 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.