

# **Achieving Language Independence with PARAGON**

*Pinki Mallick, Matt Francis, Vemulapally  
Chandraskhar, Anthony Austin, H. Alan Mantooth*

October 8th, 2003

**Electrical Engineering Department  
University of Arkansas  
Fayetteville, AR 72701**

# Introduction

- Complex systems require mixed-level modeling and simulation for effective verification and design exploration
- Designers from different scientific backgrounds don't use the same analysis tools or same hardware description languages
- Modeling in multiple hardware description languages can be both time consuming and error prone

# What is PARAGON ?

PARAGON is a modeling environment that

- promotes the creation of HDL-based models at an abstract and language-independent level
- encapsulates the semantic elements necessary to create models from mixed-signal and mixed-technology behavioral models down to semiconductor device models
- generates multiple hardware description languages like VHDL-AMS, MAST, Verilog-A(MS) from a higher-level description of the model

# Modeling Strategy in PARAGON

Enter Public Interface of Model

Generate Model Symbol

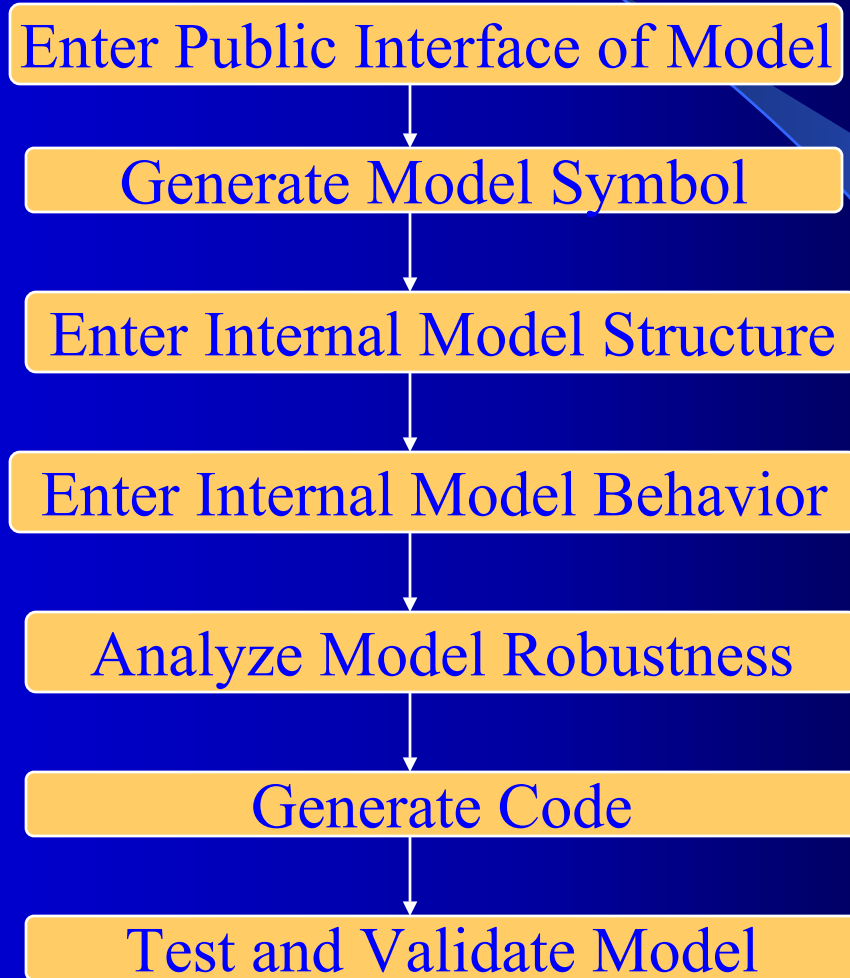
Enter Internal Model Structure

Enter Internal Model Behavior

Analyze Model Robustness

Generate Code

Test and Validate Model



# Internal model representation in PARAGON – XML Schema

The language independent internal model representation in PARAGON utilizes XML and MathML because

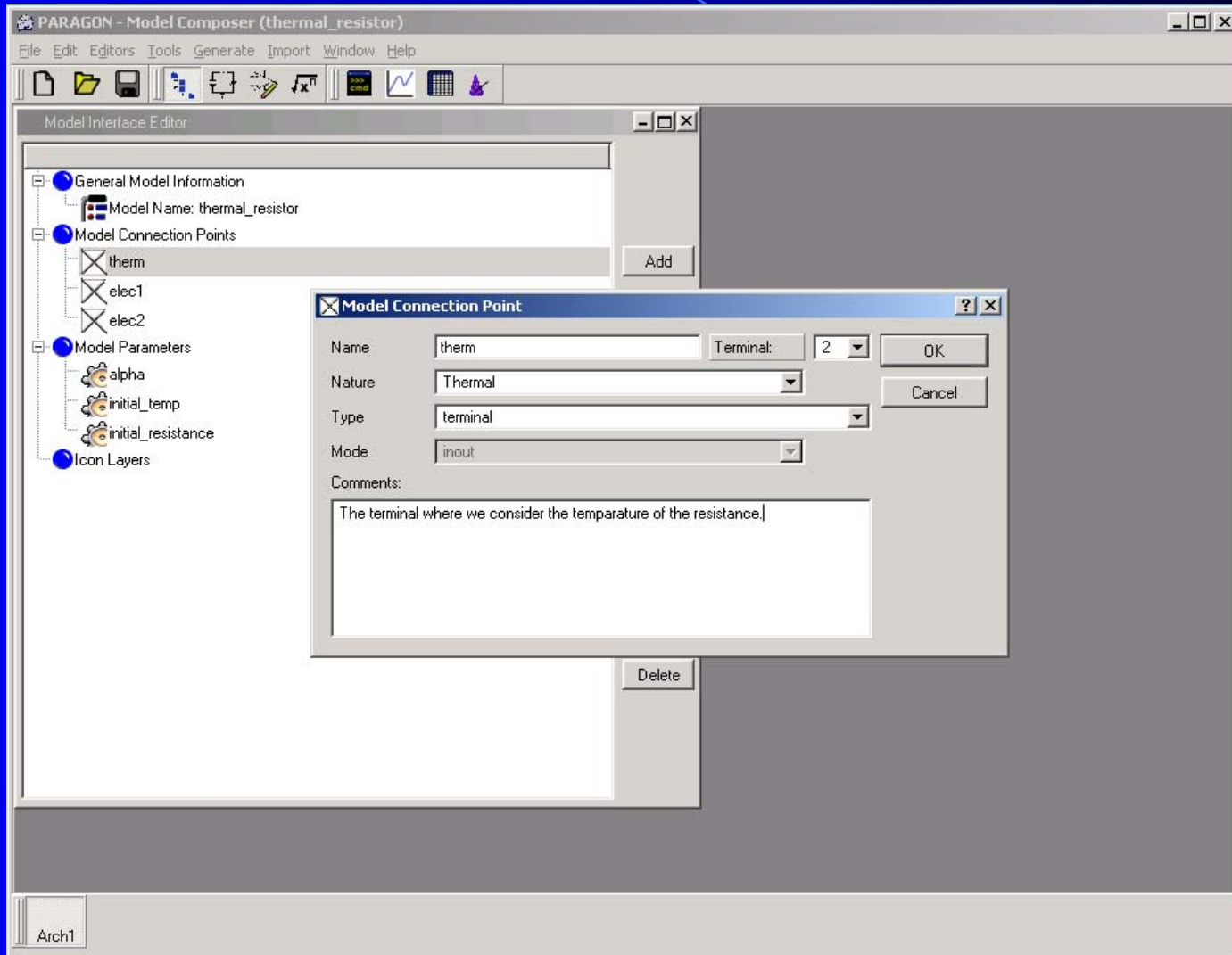
- **XML enables the description of model information in a simple and flexible structured text format**
- **XML lends itself to standardization and open sourcing for expressing model descriptions**
- **The technologies and applications built on XML provide a powerful way of expressing all types of data**
  - **MathML – XML application for describing mathematical notation**
  - **Scalable Vector Graphics(SVG) – XML application for describing two-dimensional graphics**

# Code Generation in PARAGON

The Code Generation module generates code in multiple HDLs by analyzing the internal XML model representation. Some important analysis methods of the XML Schema are:

- **Creation of an Abstract Syntax Tree (AST) by parsing the MathML expression trees**
- **Analysis of the AST for determining the functional dependency and time dependency characteristics**
- **Checking for discontinuities in model expressions and generating appropriate code**
- **Checking the ranges of validity of model parameters and generating user-friendly ‘bullet – proofed’ code**

# Illustration of Language Independence



# Illustration of Language Independence (2)

The screenshot displays the PARAGON Model Composer interface for a model named 'thermal\_resistor'. The interface is divided into several panes:

- Model Interface Editor:** Shows a tree view of the model structure. It includes 'General Model Information' (Model Name: thermal\_resistor), 'Model Connection Points' (therm, elec1, elec2), and 'Model Parameters' (alpha, initial\_temp, initial\_resistance).
- Topology Editor:** Shows a circuit diagram on a grid. It features an electrical branch with current 'i' and voltage 'v' across a resistor, and a thermal branch with temperature 'temperature' and heat flow 'heat\_flow'. The electrical branch is connected to terminals 'elec1' and 'elec2', and the thermal branch is connected to 'therm' and 'Q\_t'. Red circles highlight the electrical and thermal components, with red arrows pointing to the corresponding equations in the Equation Editor.
- Equation Editor:** Contains the following model expressions:

```
resistance=initial_resistance*(1.0+alpha*(temperature-(initial_temperature+273.0)))  
  
elec branch  
v=i*resistance  
  
thermal branch  
hflow=-(i^2)*resistance
```
- Symbol Editor:** Shows a simplified schematic of the thermal resistor model, with terminals 'elec1', 'elec2', and 'therm', and a central component labeled 'thermal resistor'.

The bottom left corner of the interface shows a tab labeled 'Arch1'.

# Internal model representation in XML and MathML

```
<model name="thermal_resistor">
```

```
<interface>
```

```
<parameter default="0.0068" name="alpha" nature="Real"  
  unit="" />
```

```
<parameter default="0.5" name="initial_resistance"  
  nature="Real" unit="" />
```

```
<parameter default="27" name="initial_temp"  
  nature="Real" unit="" />
```

```
<port mode="inout" name="elec1" nature="Electrical"  
  type="terminal" />
```

```
<port mode="inout" name="elec2" nature="Electrical"  
  type="terminal" />
```

```
<port mode="inout" name="therm" nature="Thermal"  
  type="terminal" />
```

```
</interface>
```

# Internal model representation in XML and MathML (2)

```
<body>
```

```
<branch from="elec1" name="elec_branch" to="elec2">
```

```
<quantity name="i" nature="through" type="current"  
  unit="ampere" />
```

```
<quantity name="v" nature="across" type="voltage"  
  unit="volt" />
```

```
<equation type="simultaneous">
```

```
<mrow>
```

```
<mi>v</mi>
```

```
<mo>=</mo>
```

```
<mi>i</mi>
```

```
<mo>*</mo>
```

```
<mi>resistance</mi>
```

```
</mrow>
```

```
</equation>
```

```
</branch> ..... </body></model>
```

# Code Generation – VHDL-AMS

```
library IEEE;  
library IEEE_proposed;  
use IEEE.math_real.all;  
use IEEE_proposed.electrical_systems.all;  
use IEEE_proposed.thermal_systems.all;  
  
entity thermal_resistor is  
generic (alpha:real:=0.0068;initial_resistance:real:=0.5;initial_temp:real:=27.0);
```

# Code Generation – VHDL-AMS(2)

```
port (terminal elec1:electrical;  
      terminal elec2:electrical;  
      terminal therm: thermal);  
end entity thermal_resistor;
```

```
architecture Arch1 of thermal_resistor is  
  quantity resistance : real := 1.0;  
  quantity i through elec1 to elec2;  
  quantity v across elec1 to elec2;
```

# Code Generation – VHDL-AMS(3)

```
quantity heat_flow through therm to
  thermal_ref;

quantity temperature across therm to
  thermal_ref;

begin

resistance==(initial_resistance*(1.0+(alpha
  *(temperature(initial_temp+273.0)))));

v==(i*resistance);

heat_flow==-((i**2.0)*resistance);

end architecture Arch1;
```

# Code Generation – Verilog-A

```
`include "discipline.h"  
`include "constants.h"
```

```
module thermal_resistor(elec1,elec2,therm);  
  inout elec1, elec2, therm;  
  electrical elec1, elec2;  
  thermal therm;
```

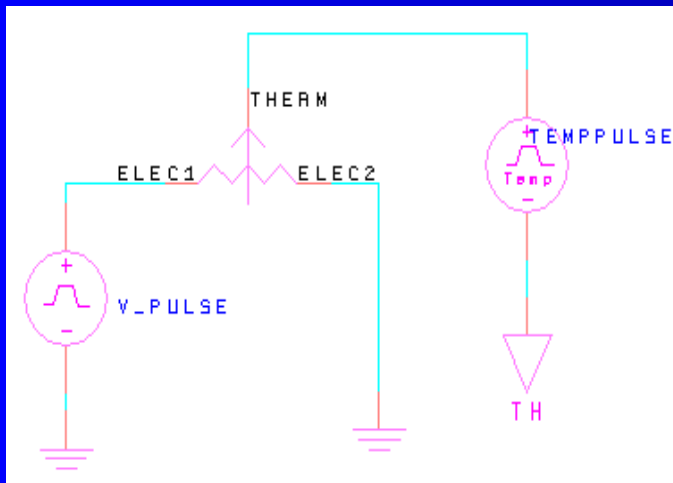
```
  parameter real alpha=0.0068;  
  parameter real initial_resistance=0.5;
```

# Code Generation – Verilog-A (2)

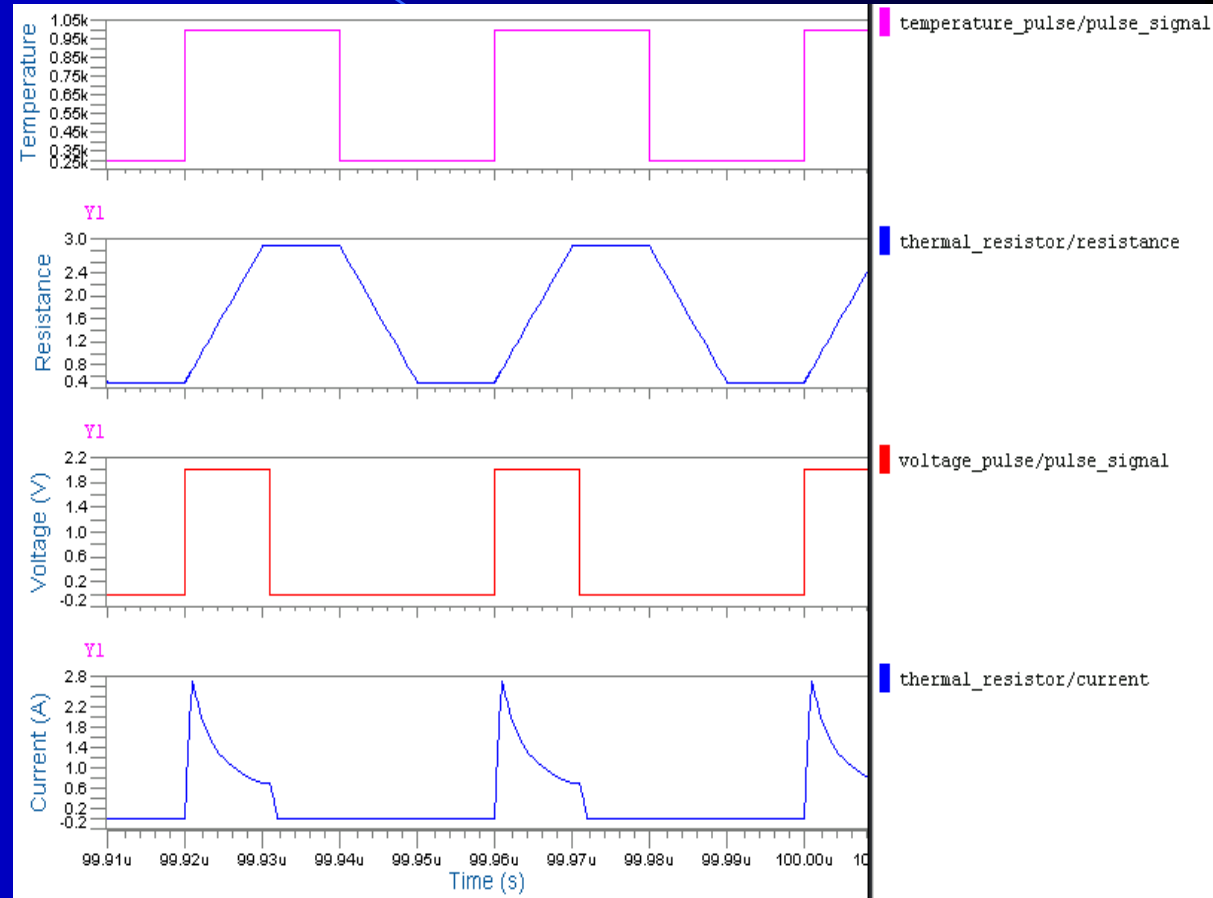
```
parameter real initial_temp=27.0;
real resistance;

analog begin
resistance=(initial_resistance*(1.0+(alpha*
    (Temp(therm)-(initial_temp+273.0)))));
V(elec1,elec2) <+ resistance*I(elec1,elec2);
Pwr(therm) <+ -
    (pow(I(elec1,elec2),2.0))*resistance;
end
endmodule
```

# Simulation Results



Testbench



Output Waveforms

# Conclusion

- PARAGON raises model creation and support above programming in hardware description language
- Knowledge of basic modeling concepts and mathematics is the only requirement for creating models using PARAGON
- PARAGON can be downloaded from <http://mixedsignal.eleg.uark.edu>

# QUESTIONS ?

