ANALOG MODELING WITH VERILOG-A USING CADENCE TOOLS

Eng. Sherief Fathi

GUC
German University in Cairo

Hardware Description Language (HDL)

- In electronics, a hardware description language (HDL) is a ***specialized computer language*** used to ***describe the structure and behavior of electronic circuits.***

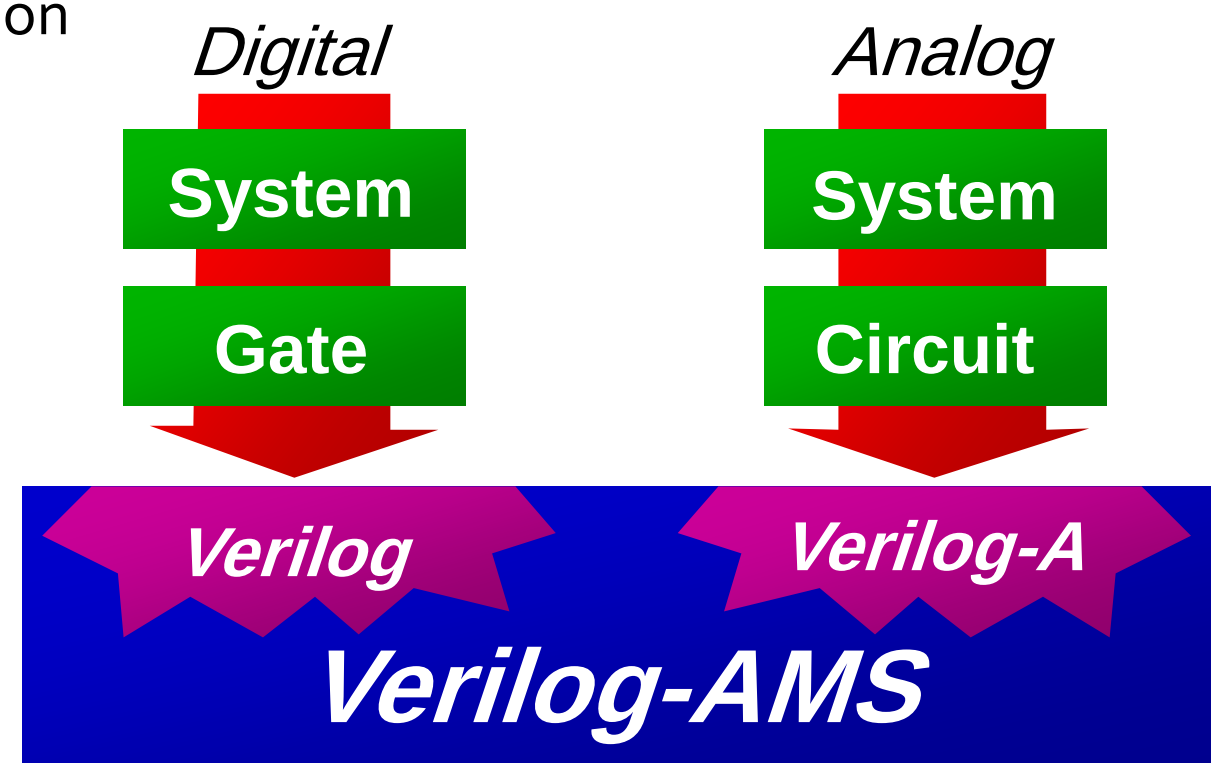
- Two HDLs used today:

VHDL and Verilog

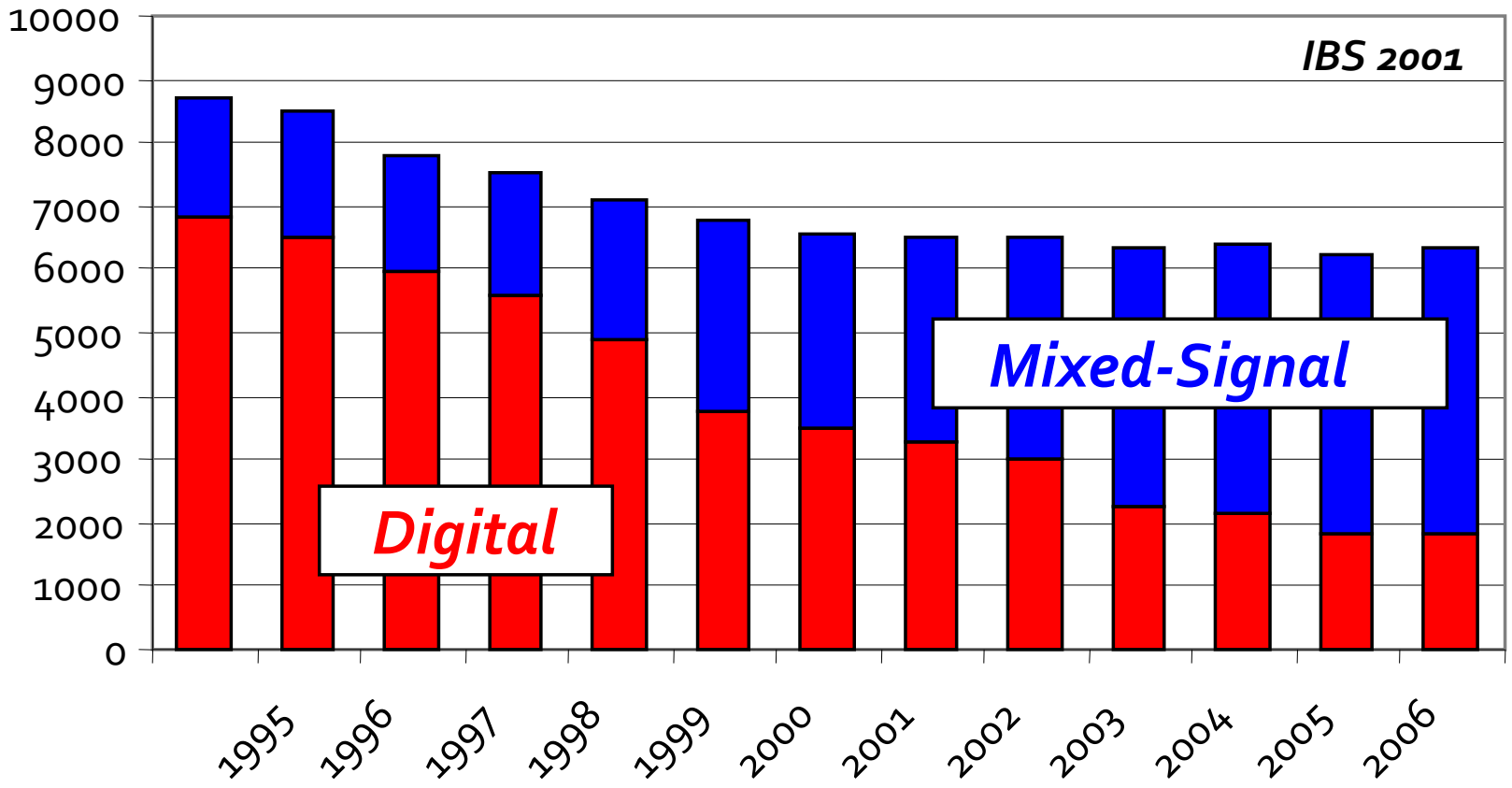
Both are industrial standards and are supported by most software tools.

Verilog-AMS

- **Combines Verilog**, …
  Discrete-event / discrete-value simulation
- **Verilog-A**, …
  - Continuous-time / continuous-value simulation
  - Signal flow modeling
  - Conservative modeling
- ***And some extras***
  - Discrete-event / continuous value simulation
  - Automatic interface element insertion



Mixed-Level Simulation (MLS)



Verilog-A

- **Able to model analog circuits and systems**

  - Signal-flow models

  Model relates potentials only

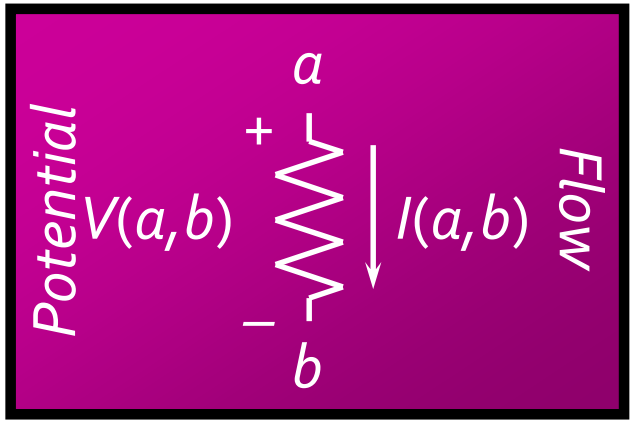
  Useful for abstract models

  - Conservative models

  Model relates potentials and flows

  Device modeling and loading at interfaces

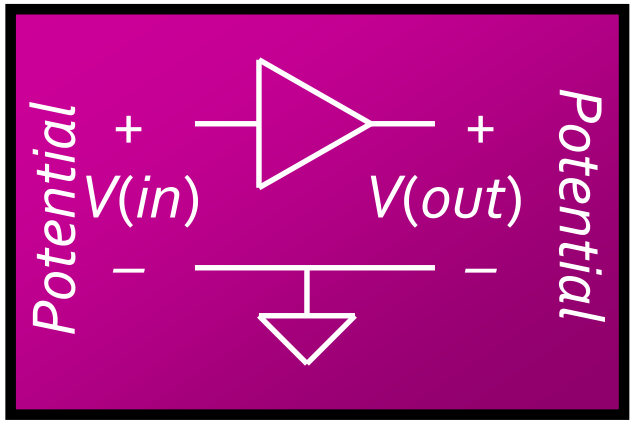
Potentials and Flows



Resistor

```
module resistor (a, b);
    electrical a, b;
    parameter r = 1;

    analog
        V(a,b) <+ r*I(a,b);
endmodule
```

Conservative Model (potential & flow)



Amplifier

```
module amp (out, in);
    output out; input in;
    voltage out, in;
    parameter a = 1;

    analog
        V(out) <+ a*V(in);
endmodule
```

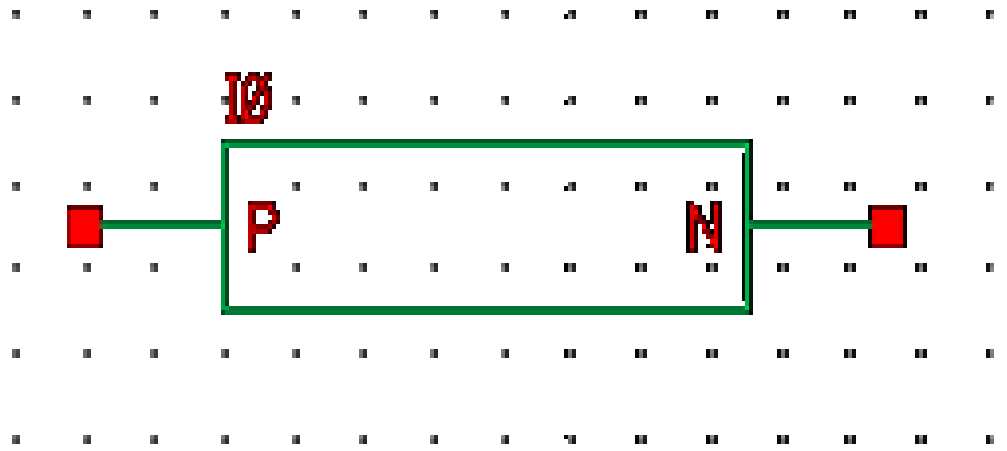
Signal-Flow Model (potential only)

Declaring Modules

module mymod(p,n);

.....

endmodule



Ports

```
module modName(outPort1, outPort2, inPort, bidirectional);
   output outPort1;
   output [3:0] outPort2;
   input inPort;
   inout bidirectional;
   electrical outPort1, [3:0] outport2, inPort, bidirectional;
   ……
endmodule
```

Constants

- **parameter real** res = 5e3 from (0:inf);
- **parameter real** percentage = 0.7 from [0:1];
- **parameter integer** N = 4 from (0:inf);
- **parameter real** poles[0:3] = { 1.0, 3.198, 4.554, 2.00 };
- **parameter string** transistortype = "NPN" from { "NPN", "PNP" };

Variables

- **integer** a[1:64]; // an array of 64 integer values
- **real** float; // a variable to store real value
- **real** gain_factor[1:30]; // array of 30 gain multipliers

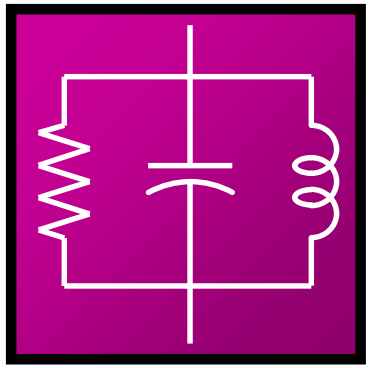
Signal Access on Nodes, Ports, Branches

- To **get (probe) values**: use access function in expression
  - *x* gets voltage between *a* and ground: $x = V(a)$
  - *x* gets voltage between *a* and *b*: $x = V(a,b)$
  - *x* gets current between *a* and ground: $x = I(a)$
  - *x* gets current between *a* and *b*: $x = I(a,b)$
- To **set (source) values**: use access function as target in contribution
  - set voltage between *a* and ground to *x*: $V(a) <+ x$
  - set voltage between *a* and *b* to *x*: $V(a,b) <+ x$
  - set current between *a* and ground to *x*: $I(a) <+ x$
  - set current between *a* and *b* to *x*: $I(a,b) <+ x$

The Contribution Operator : '<+'

- *Accumulates potentials or flows to nodes, ports, and branches*
- Order of contribution is not significant

```
module rlc (a, b);
    electrical a, b;
    parameter R = 1 exclude 0;
    parameter C = 1;
    parameter L = 1 exclude 0;

    analog begin
        I(a,b) <+ V(a,b) / R;
        I(a,b) <+ C*ddt(V(a,b));
        I(a,b) <+ idt(V(a,b) / L;
    end
endmodule
```



Assignment vs Contribution

- **Example1:**

```
module test(p,n) ;
inout  p,n;        electrical p,n;
parameter integer a = 2;        real r;
analog        begin
```

```
r=a;    // Now r=2
r=2*a;  // The value of r is updated to r=4
V(n) <+ a;    // This equation contributes by a value a ,    V(n) =a=2;
V(n) <+ 2*a;   // A new contribution is added to V(n)  ,      V(n) = a+2a =6;
```

```
end                endmodule
```

Operators

Arithmetic Operators	+, -, *,**, /, %
Relational Operators	<, <=, >, >=
Equality Operators	==, !=, ===, !==
Logical Operators	!, &&, ||
Bit-Wise Operators	~, &, |, ^, ~^
Unary Reduction	&, ~&, |, ~|, ^, ~^
Shift Operators	>>, <<
Conditional Operators	?:
Concatenations	{}

Analog Operators

- **Differentiator: ddt()**
  Time derivative of its argument
- **Integrator: idt()**
  Time integral of its argument
  Optional initial condition
- **Circular integrator: idtmod()**
  Time integral of its argument passed through modulus operation
- **Time delay: absdelay()**
  Delayed version of its argument

Standard mathematical functions

Verilog function style	Traditional Verilog-AMS function style	Equivalent C function	Description	Domain
`$ln`(x)	`ln`(x)	`log`(x)	Natural logarithm	$x > 0$
`$log10`(x)	`log`(x)	`log10`(x)	Decimal logarithm	$x > 0$
`$exp`(x)	`exp`(x)	`exp`(x)	Exponential	All x
`$sqrt`(x)	`sqrt`(x)	`sqrt`(x)	Square root	$x >= 0$
-	`min`(x, y)	-	Minimum	All x, all y
-	`max`(x, y)	-	Maximum	All x, all y
-	`abs`(x)	-	Absolute	All x
`$pow`(x,y)	`pow`(x, y)	`pow`(x,y)	Power (x^y)	if $x > 0$, all y, if $x = 0$, $y > 0$; if $x < 0$, `int`(y)
`$floor`(x)	`floor`(x)	`floor`(x)	Floor	All x
`$ceil`(x)	`ceil`(x)	`ceil`(x)	Ceiling	All x

Trigonometric and hyperbolic functions

Verilog function style	Traditional Verilog-AMS function style	Equivalent C function	Description	Domain
$sin(*x*)	**sin**(*x*)	**sin**(*x*)	Sine	All *x*
$cos(*x*)	**cos**(*x*)	**cos**(*x*)	Cosine	All *x*
$tan(*x*)	**tan**(*x*)	**tan**(*x*)	Tangent	*x* != *n* (π / 2), *n* is odd
$asin(*x*)	**asin**(*x*)	**asin**(*x*)	Arc-sine	-1 <= *x* <= 1
$acos(*x*)	**acos**(*x*)	**acos**(*x*)	Arc-cosine	-1 <= *x* <= 1
$atan(*x*)	**atan**(*x*)	**atan**(*x*)	Arc-tangent	All *x*
$atan2(y,*x*)	**atan2**(*y*,*x*)	**atan2**(*y*,*x*)	Arc-tangent of *y*/*x*	All *x*, all *y*; atan2(0,0) = 0
$hypot(*x*,*y*)	**hypot**(*x*,*y*)	**hypot**(*x*,*y*)	$\sqrt{x^2 + y^2}$	All *x*, all *y*
$sinh(*x*)	**sinh**(*x*)	**sinh**(*x*)	Hyperbolic sine	All *x*
$cosh(*x*)	**cosh**(*x*)	**cosh**(*x*)	Hyperbolic cosine	All *x*
$tanh(*x*)	**tanh**(*x*)	**tanh**(*x*)	Hyperbolic tangent	All *x*
$asinh(*x*)	**asinh**(*x*)	**asinh**(*x*)	Arc-hyperbolic sine	All *x*
$acosh(*x*)	**acosh**(*x*)	**acosh**(*x*)	Arc-hyperbolic cosine	*x* >= 1
$atanh(*x*)	**atanh**(*x*)	**atanh**(*x*)	Arc-hyperbolic tangent	-1 < *x* < 1

If Statement (SWITCH)

Example 1

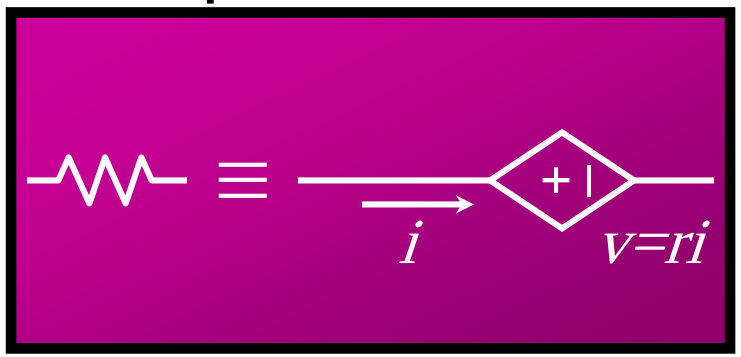
```
if (x1 >= 2)    begin
   x2= 5*x1;
end
else if (x1<5)  begin
   x2= x1/2;
end
else   begin
x2 = x1;
end
```

Example 2

```
if(analysis("ic"))     begin
   x2= init_cond;
end
else if(analysis("tran"))      begin
   x2= x1/2;
end
```

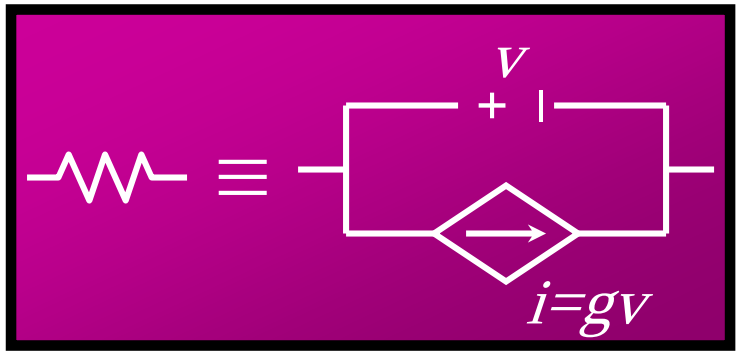
Example: Resistor and Conductor

- **Resistor (potential source branch)**




```
module res (a, b);
    electrical a, b;
    parameter real r = 1;

    analog
        V(a,b) <+ r*I(a,b);
endmodule
```

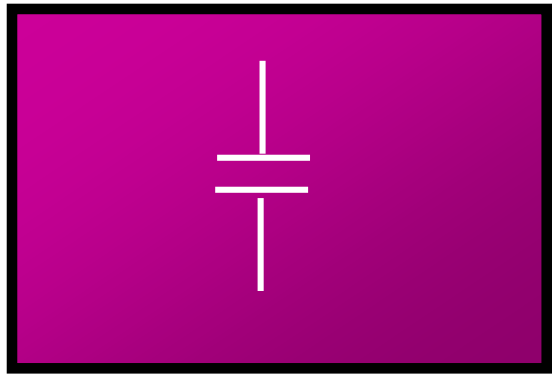
- **Conductor (flow source branch)**




```
module cond (a, b);
    electrical a, b;
    parameter real g = 1;

    analog
        I(a,b) <+ g*V(a,b);
endmodule
```

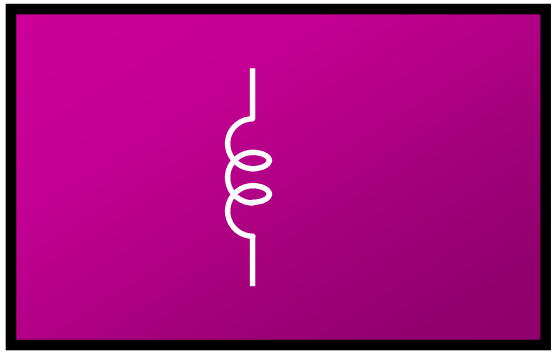
Example: Capacitor and inductor

- **Capacitor**



```
module cap (a, b);
    electrical a, b;
    parameter c = 1;

    analog
        I(a,b) <+ C*ddt(V(a,b));
endmodule
```

- **inductor**



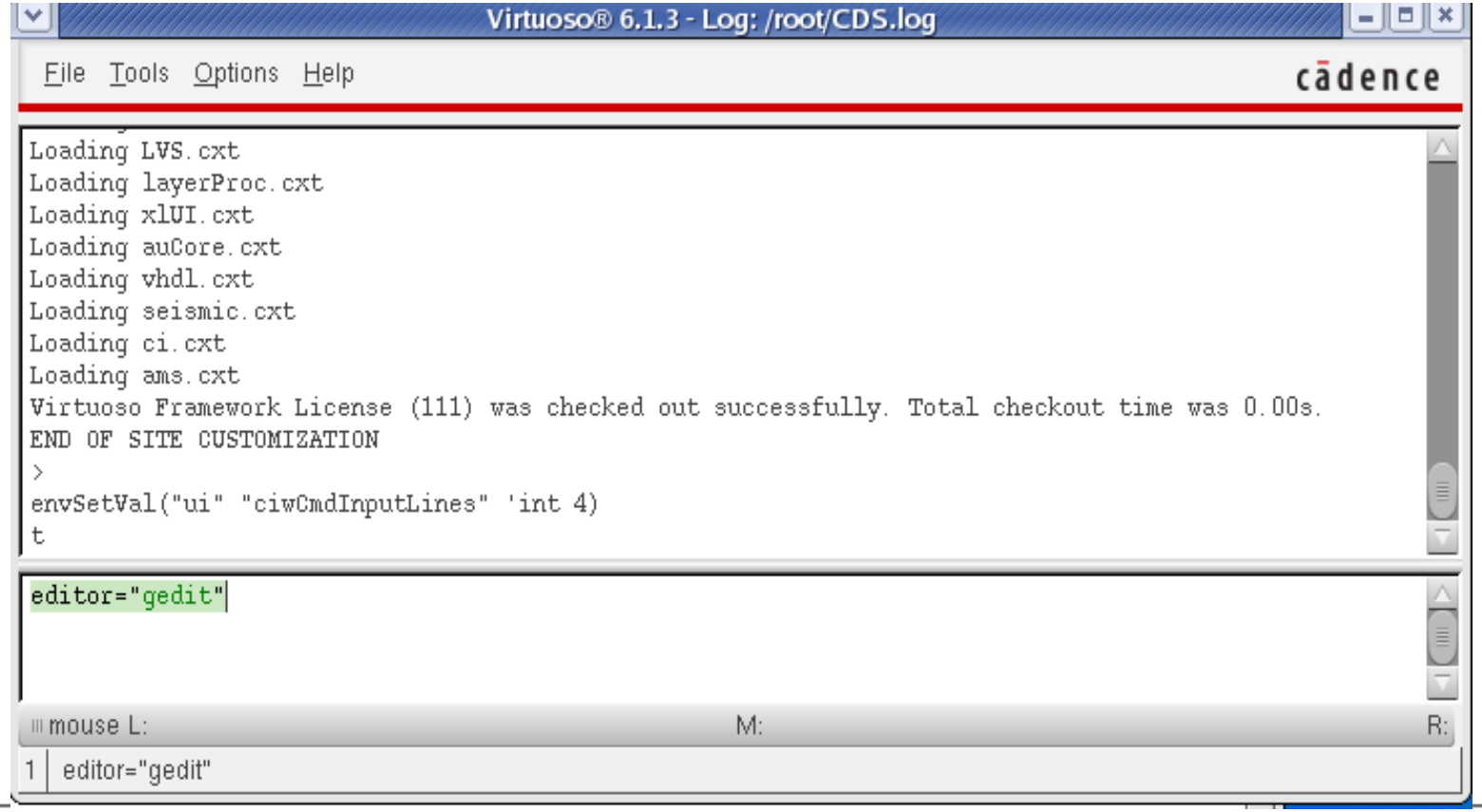
```
module ind (a, b);
    electrical a, b;
    parameter L = 1;

    analog
        V(a,b) <+ L*ddt(I(a,b));
endmodule
```

Example: Multiplier

```
//VerilogA for a multiplier,
`include "constants.vams"
`include "disciplines.vams“
module multiplier(in1,in2,out);
input in1,in2;
output out;
electrical in1,in2,out;
analog begin
V(out) <+ V(in1) * V(in2);
end
endmodule
```

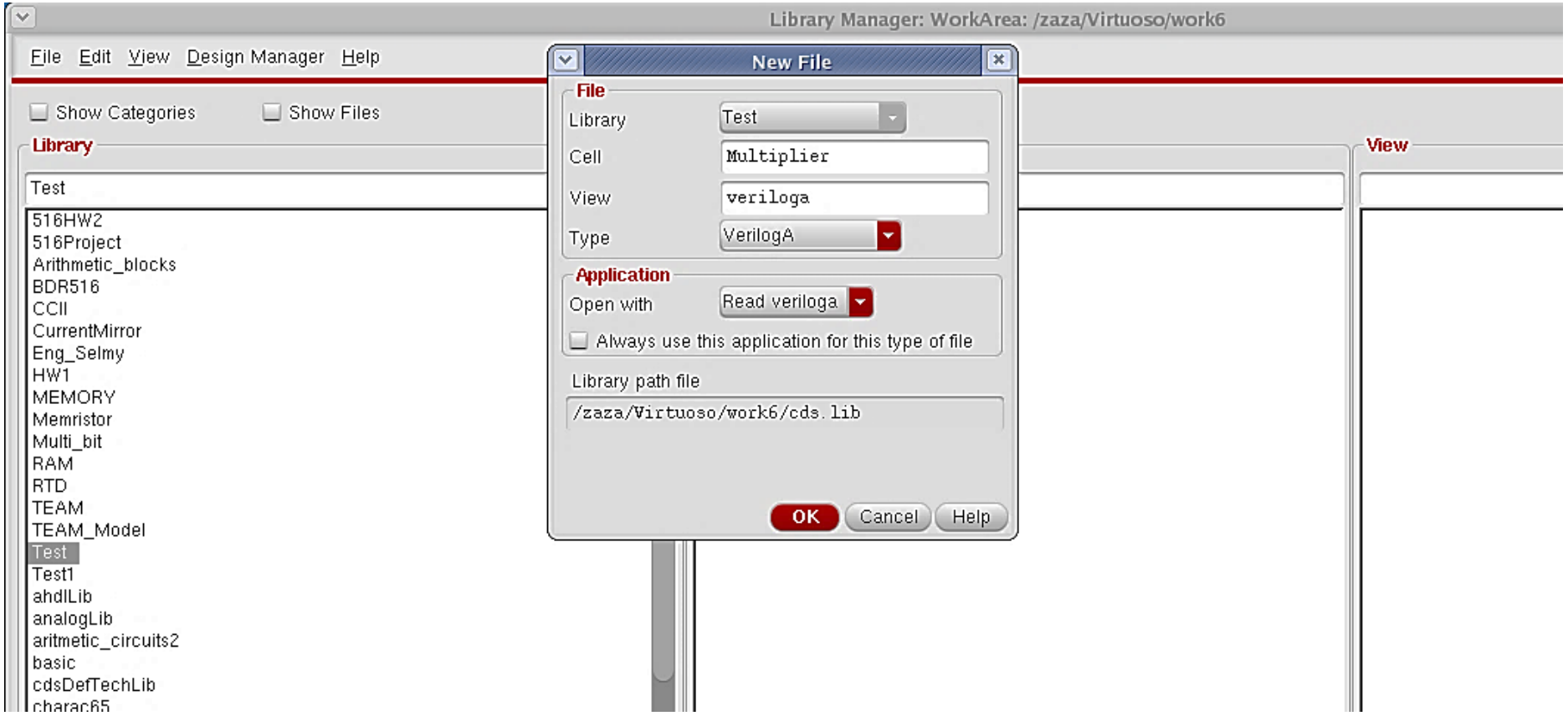
Inserting Verilog-A model into Cadence

- Open Cadence then change the editor: editor="gedit"



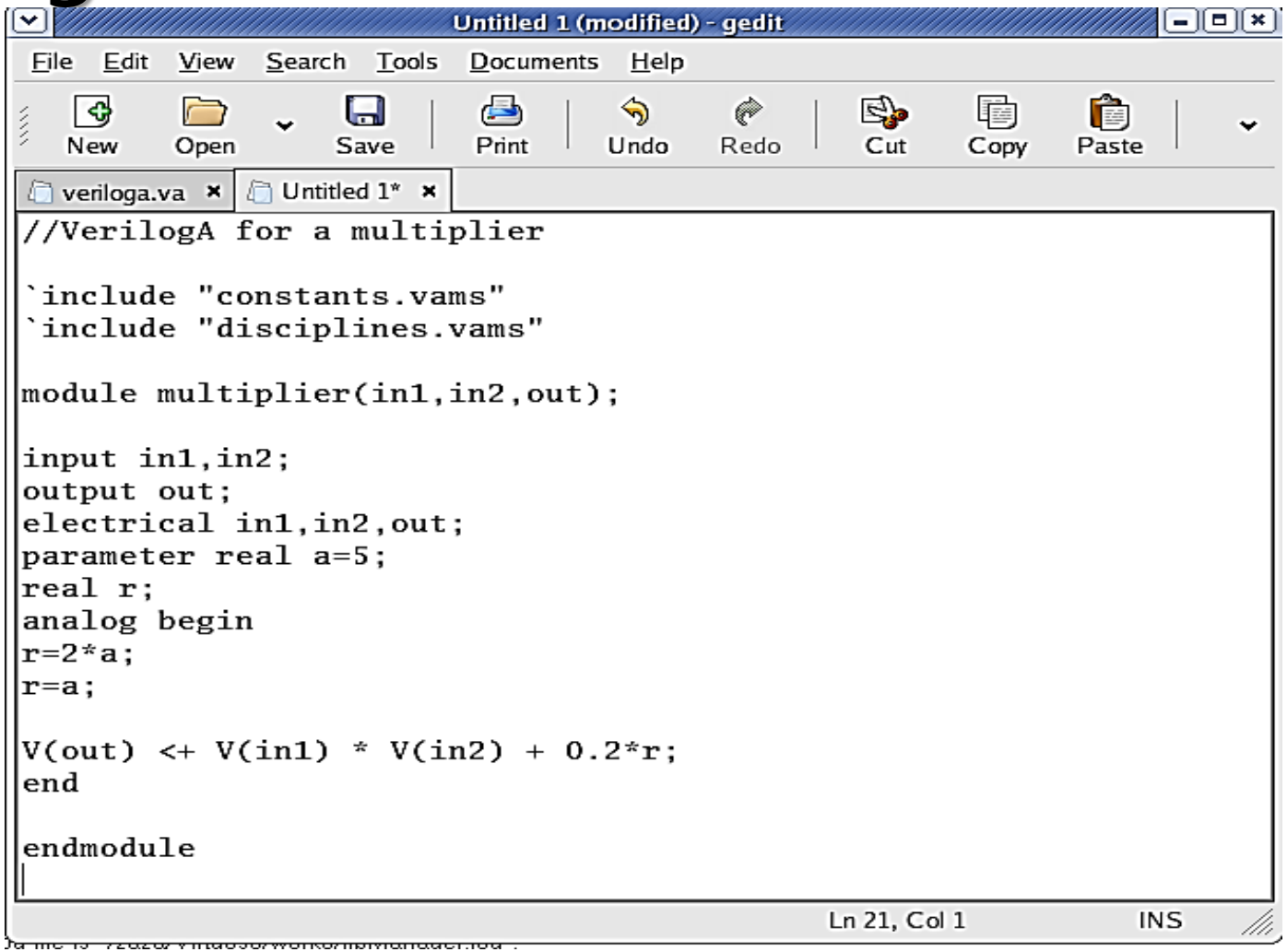
Inserting Verilog-A model into Cadence

- From the library manager, create a “new cell view” with a type: “veriloga”



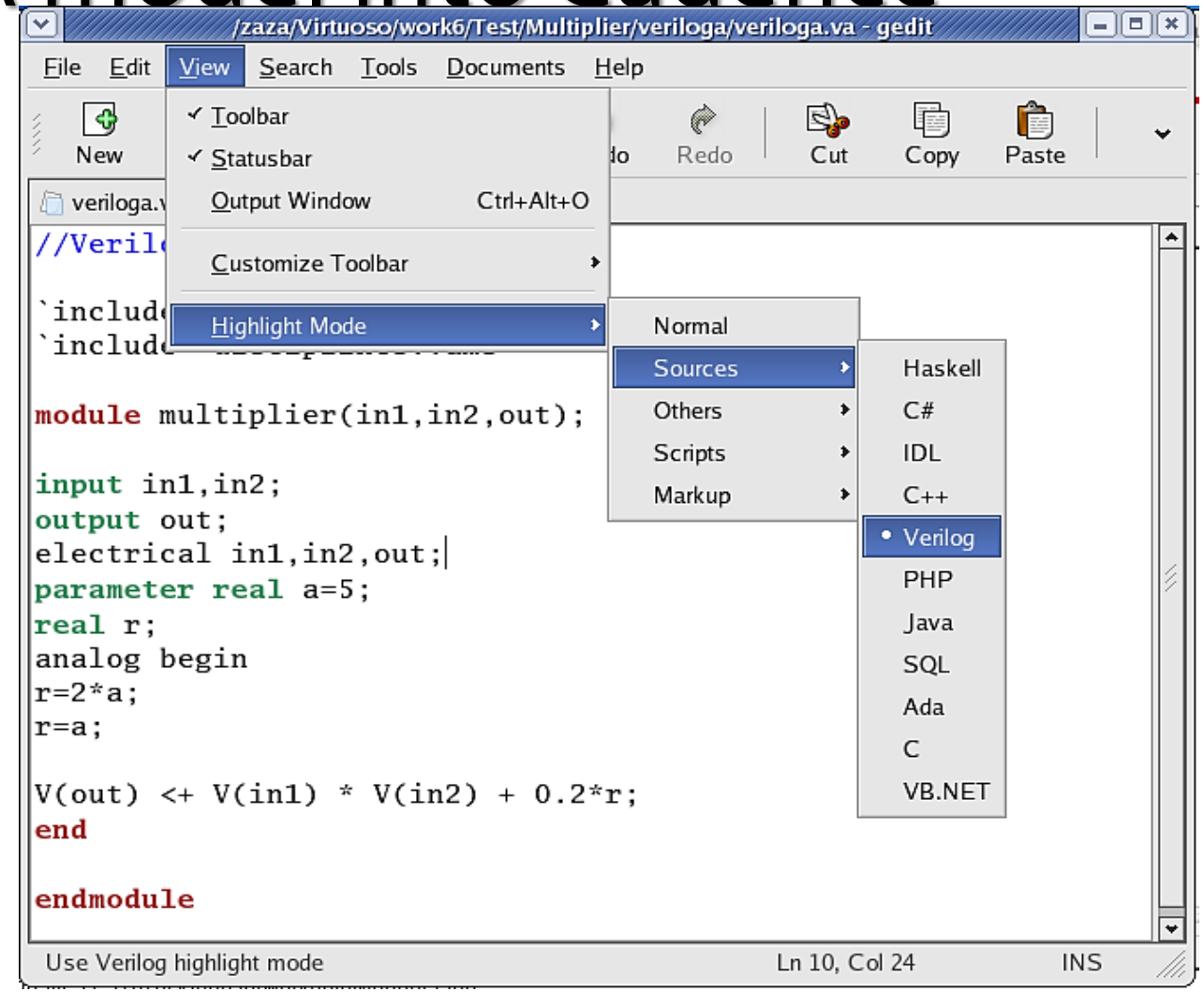
Inserting Verilog-A model into Cadence

- Insert the VerilogA code:



Inserting Verilog-A model into Cadence

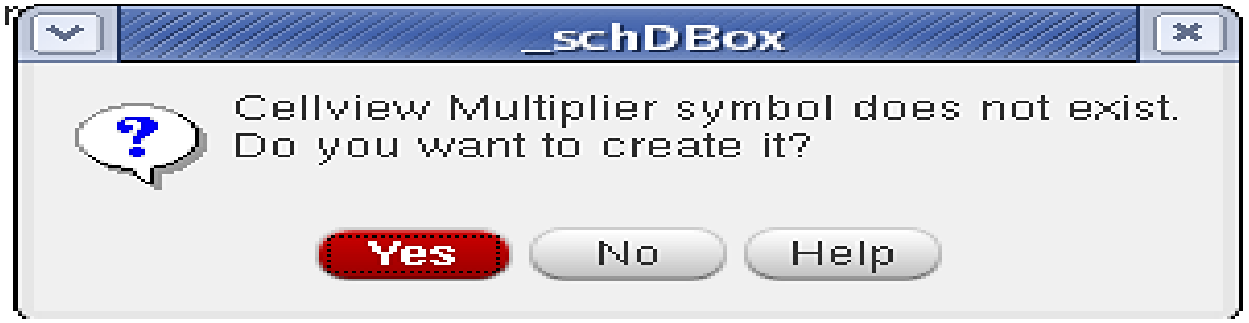
- Choose highlight mode > verilog



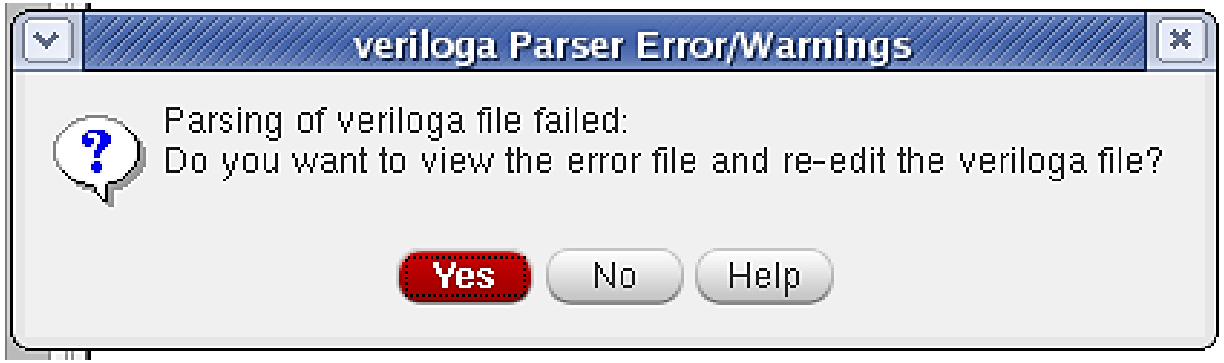
Inserting Verilog-A model into Cadence

- Insert the VerilogA code, save the file, and close.

1. If the model does not has a syntax error, you will be asked to create a symbol.

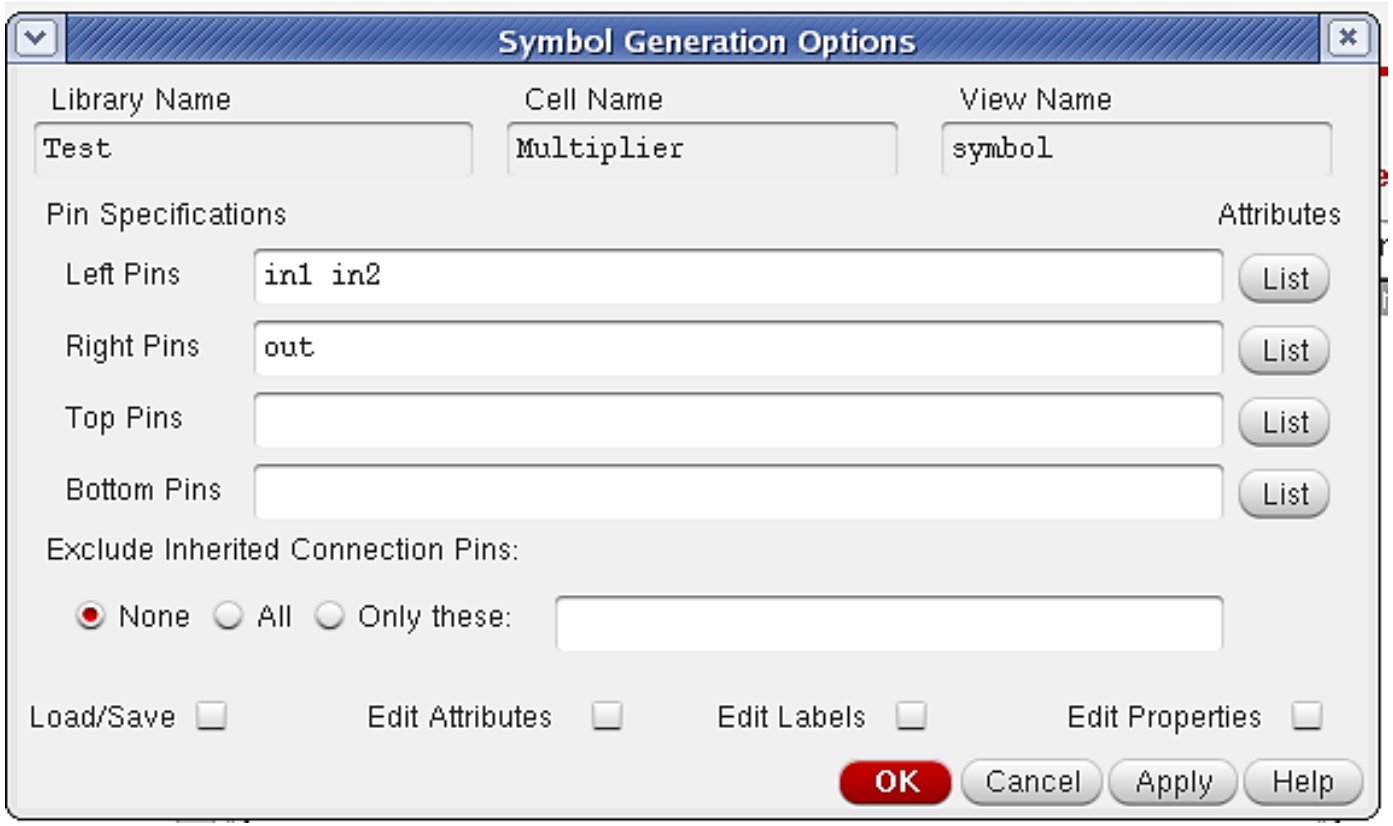


2. If the model has a syntax error, open the error log to identify errors and solve it.



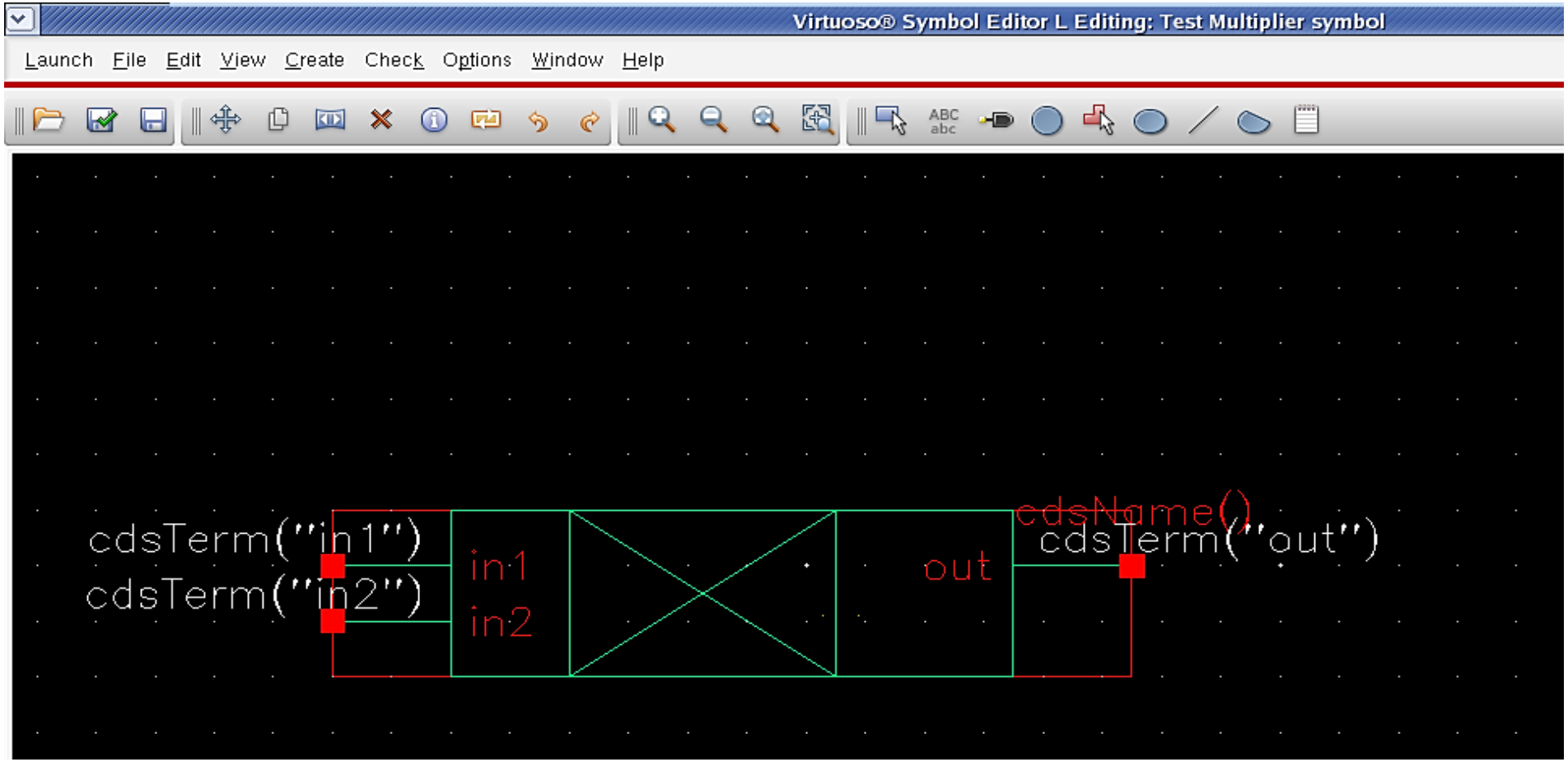
Inserting Verilog-A model into Cadence

- Define the directions of the pins i.e. : left, right, top, or bottom



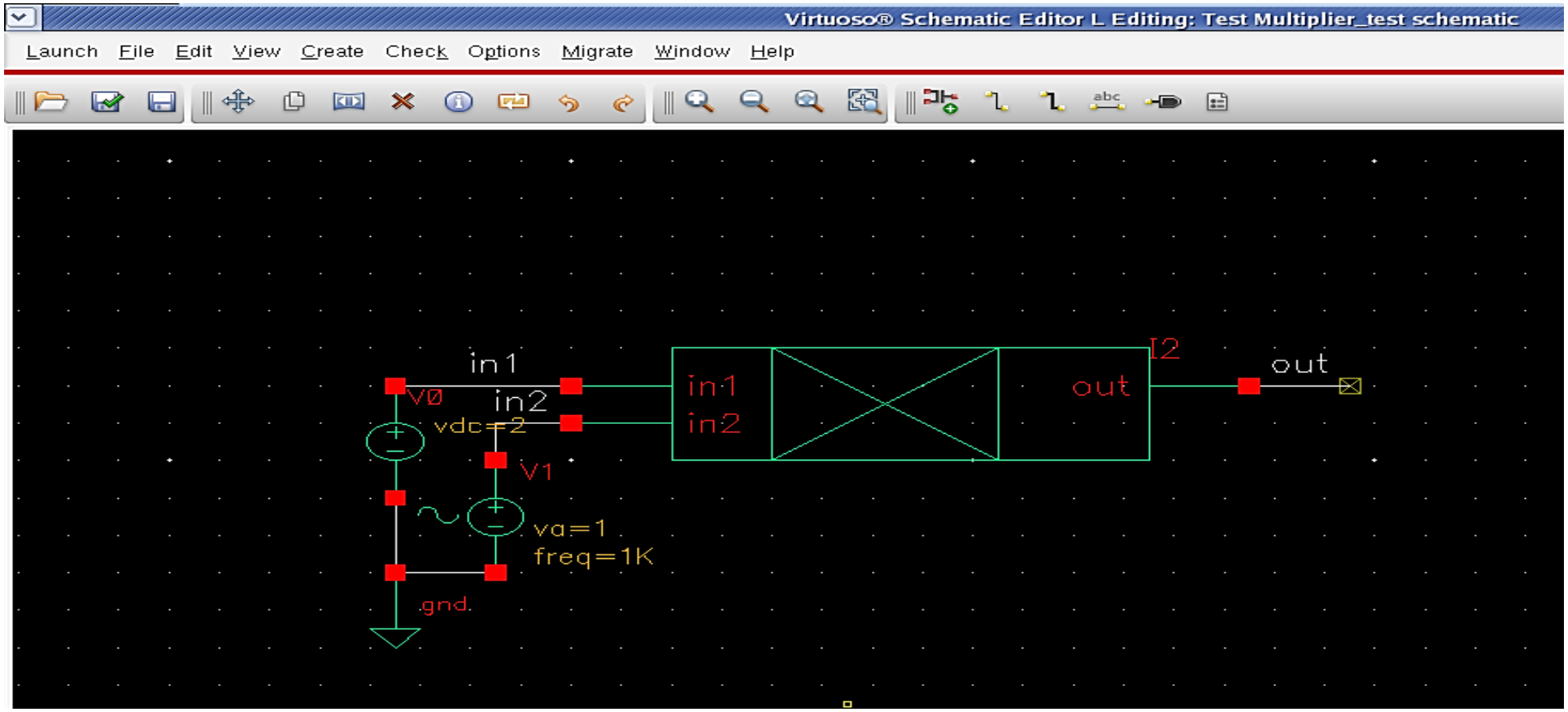
Inserting Verilog-A model into Cadence

- Modify the symbol “optional”, then Check & Save

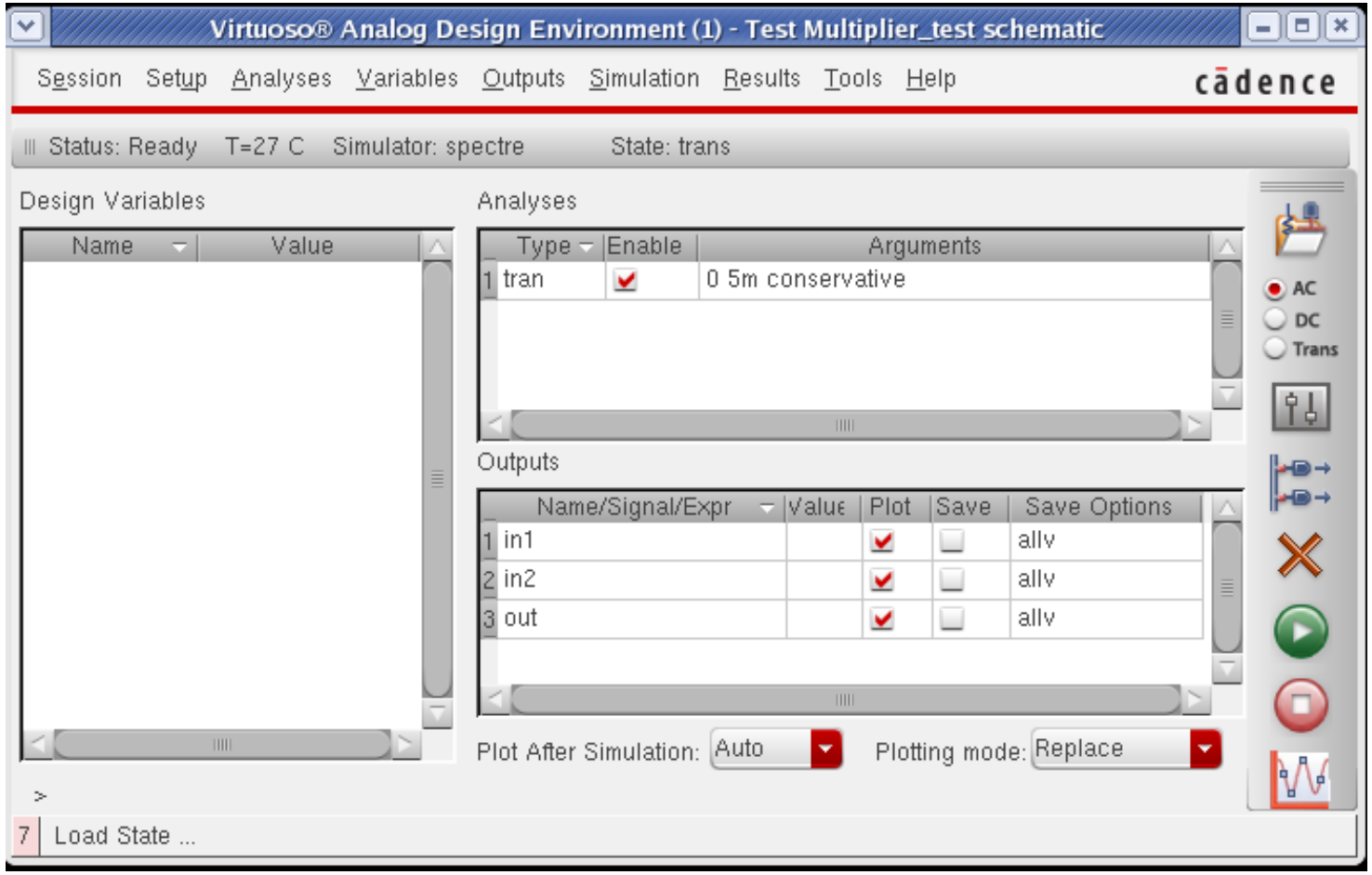


Inserting Verilog-A model into Cadence

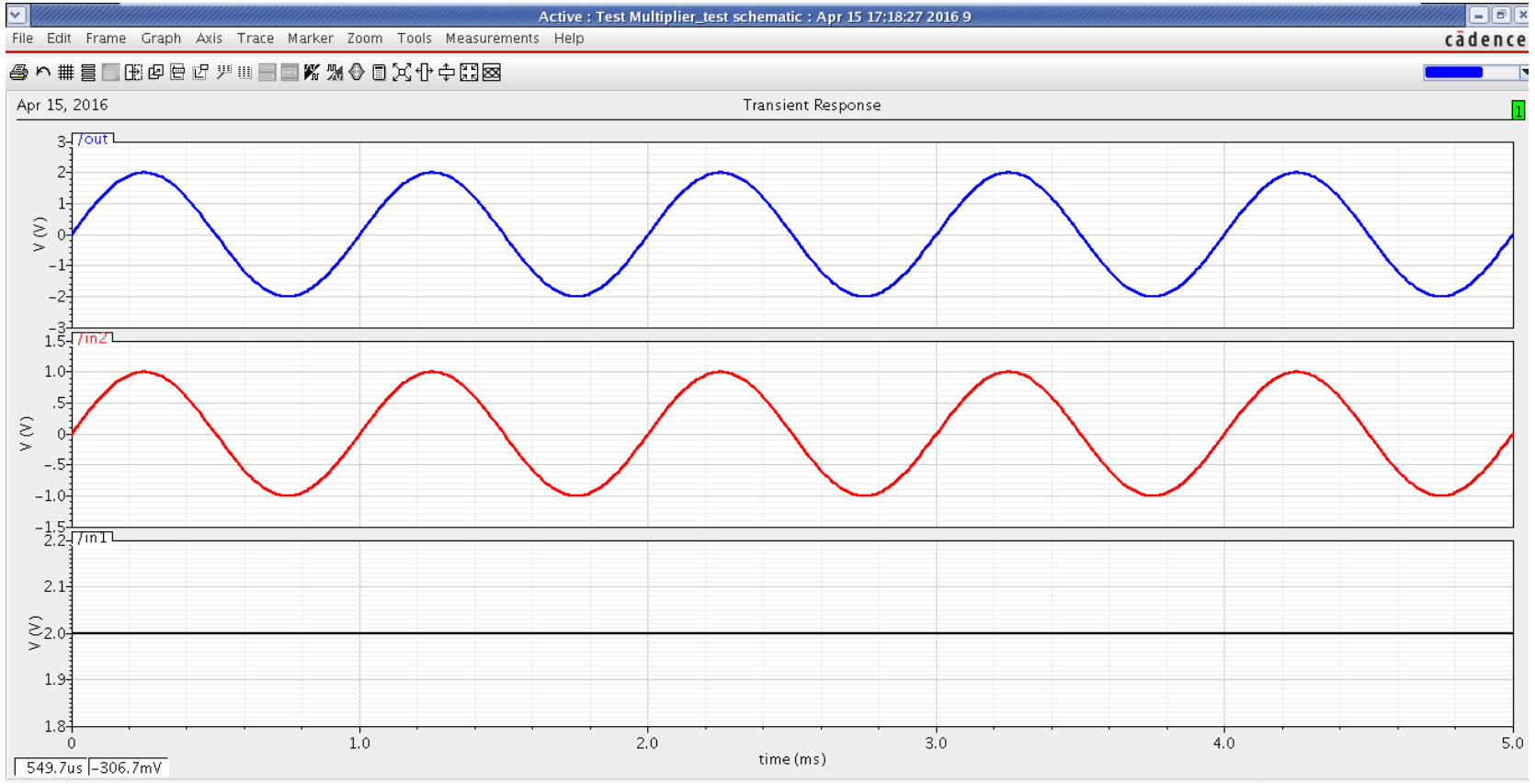
- Now you can test your model by creating a new cell view of type “Schematic”, and make a simple test circuit:



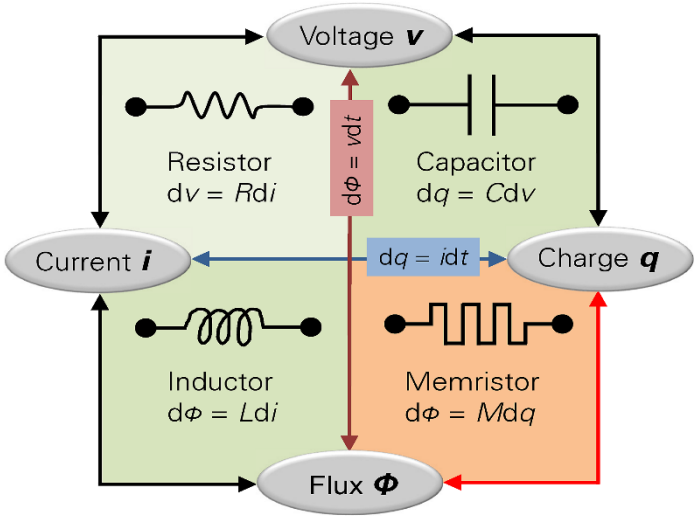
Inserting Verilog-A model into Cadence

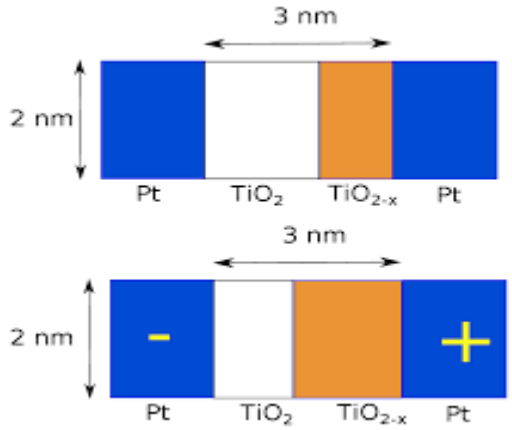


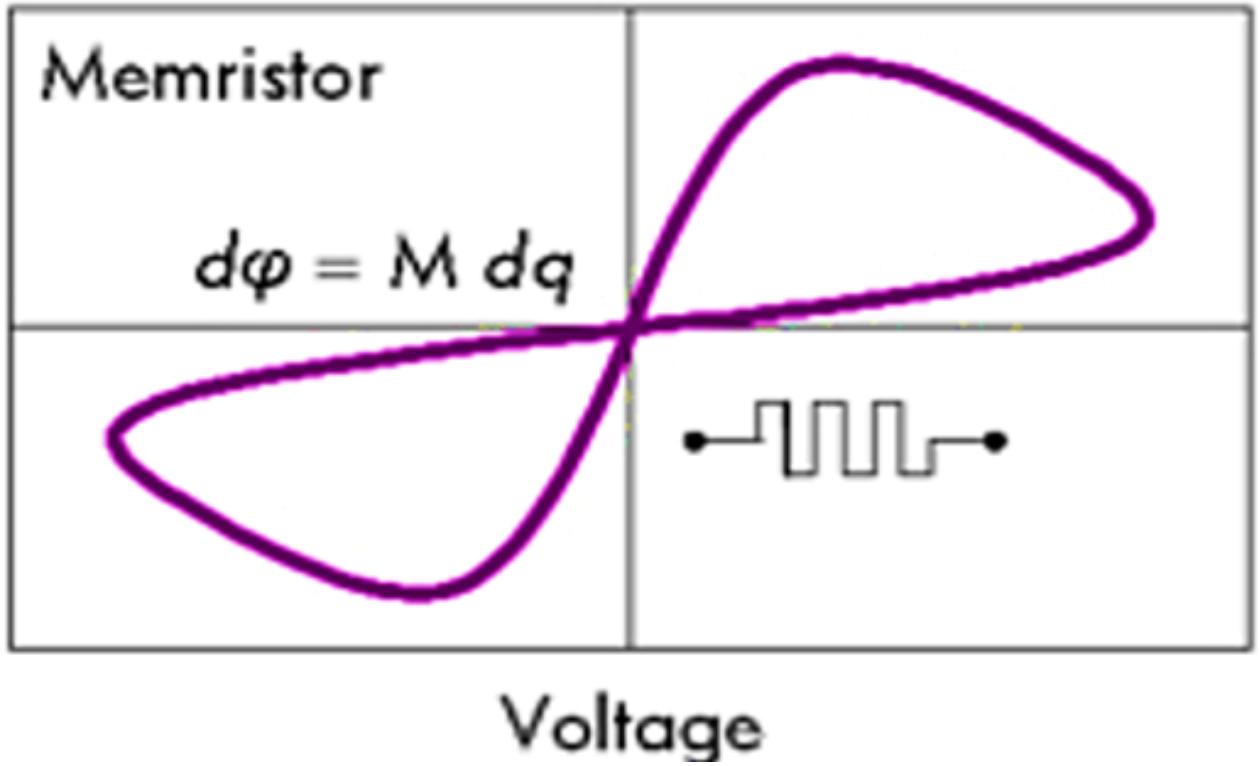
Inserting Verilog-A model into Cadence



Example B: TEAM Memristor model



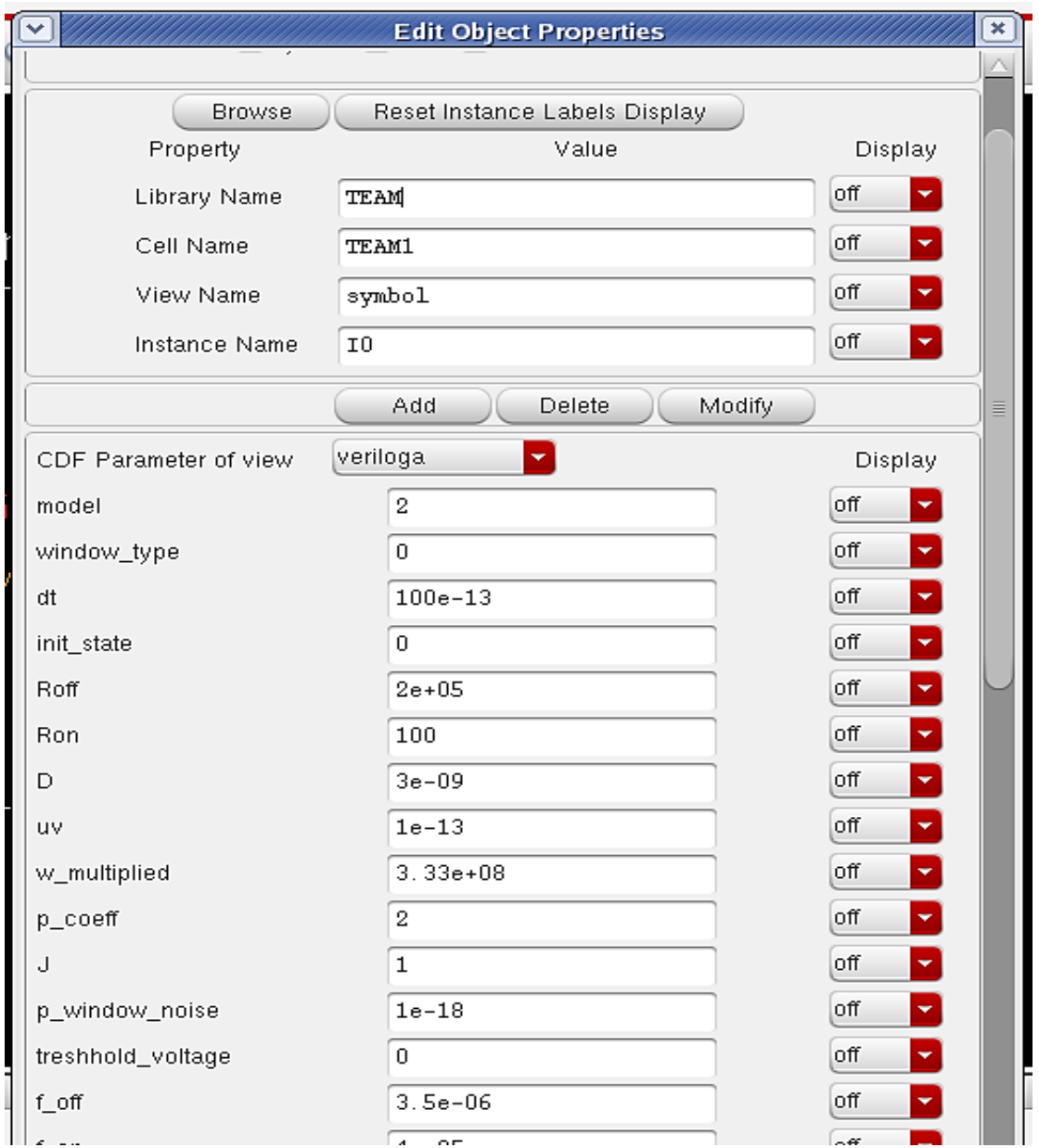




http://webee.technion.ac.il/people/skva/Memristor%20Models/TCAS_memristor_model_paper_final.pdf

Example B: TEAM Memristor model

***Note**: You can access the parameters' values from the schematic directly*



Some useful links

- Verilog-AMS Manual

  http://www.accellera.org/images/downloads/standards/v-ams/VAMS-LRM-2-3-1.pdf

- Verilog-A model library at

  http://www.designers-guide.org/VerilogAMS/

Questions

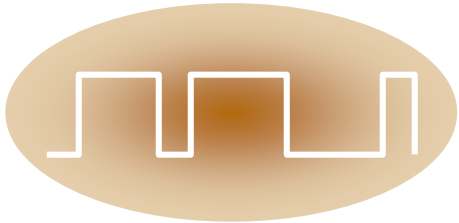
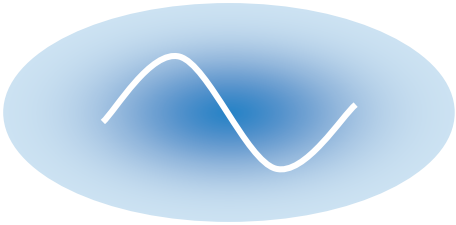
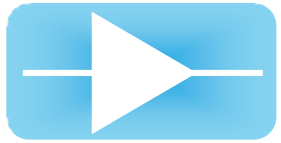
Thank you

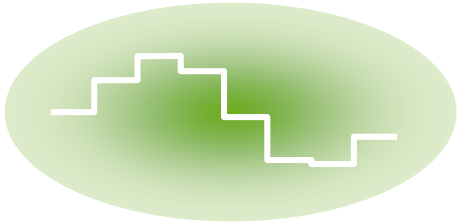
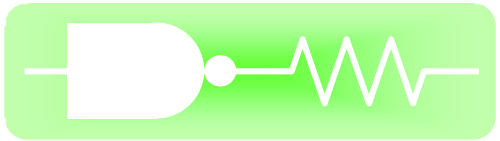
Operators

{} {{}}	Concatenation, replication
unary +, unary -	Unary operators
+ - * / **	Arithmetic
%	Modulus
> >= < <=	Relational
!	Logical negation
&&	Logical and
||	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or
^~ or ~^	Bitwise equivalence
&	Reduction and
~&	Reduction nand
|	Reduction or

Verilog-AMS

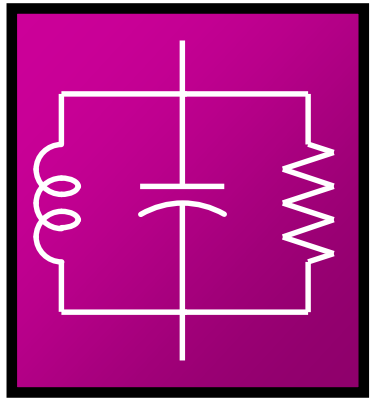
- **Combines Verilog**, …
  - Discrete-event / discrete-value simulation
- **Verilog-A**, …
  - Continuous-time / continuous-value simulation
  - Signal flow modeling
  - Conservative modeling
- ***And some extras***
  - Discrete-event / continuous value simulation
  - Automatic interface element insertion

Named Branches

- Named branches are explicitly declared
  - Useful when defining distinct *parallel potential branches*



```
module rlc (a, b);
    electrical a, b;
    parameter R = 1, C = 1, L= 1;
    branch (a, b) res, cap, ind;

    analog begin
        V(res) <+ R*I(res);
        I(cap) <+ C*ddt(V(cap));
        V(ind) <+ L*ddt(I(ind));
    end
endmodule
```

Example: Capacitor with Initial Condition

```
module cap (a, b);
    electrical a, b;
    parameter real c=0, ic=0;
 analog begin
        if (analysis("ic"))
            V(a,b) <+ ic;
        else
            I(a,b) <+ ddt(c*V(a,b));
    end
endmodule
```

Examples:

```
module adc4 (out, rem, in);
   output [3:0] out; output rem;
   input in;
   electrical [3:0] out;
   electrical in, rem, rem_chain;

   adc2 hi2 (out[3:2], rem_chain, in);
   adc2 lo2 (out[1:0], rem, rem_chain);
endmodule

module adc2 (out, remainder, in);
   output [1:0] out; output remainder;
   input in;
   electrical [1:0] out;
   electrical in, remainder, r;

   adc hi1 (out[1], r, in);
   adc lo1 (out[0], remainder, r);
endmodule
```

Accessing Net and Branch Signals

- **Examples**

```
Vin = V(in);

CurrentThruBranch = I( myBranch );
```

- **Indirect branch assignment**

An indirect branch assignment is useful when it is difficult to solve an equation. It has this format,

```
V(n) : V(p) == 0;
```

which can be read as "***find V(n) such that V(p) is equal to zero***."

Operator Precedence

[ ]	bit-select or part-select
( )	parentheses
!, ~	logical and bit-wise negation
&, |, ~&, ~|, ^, ~^, ^~	reduction operators
+, -	unary arithmetic
{ }	concatenation
*, /, %	arithmetic
+, -	arithmetic
<<, >>	shift

>, >=, <, <=	relational
==, !=	logical equality
&	bit-wise AND
^, ^~, ~^	bit-wise XOR and XNOR
|	bit-wise OR
&&	logical AND
||	logical OR
? :	conditional

Analog operators and equations

- Time derivative operator
- Time integral operator
- Circular integrator operator

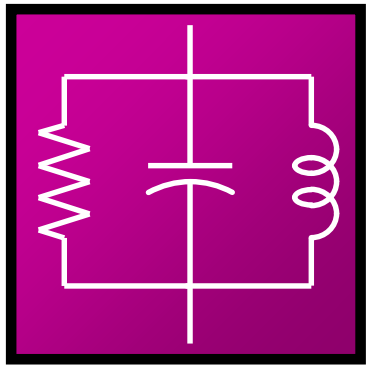
Table 4-20—Analog operator arguments

Operator	Constant expression arguments	Dynamic expression arguments
absdelay	maxdelay	expr, td
ddt	abstol	expr
ddx	wrt_what	expr
idt	abstol	expr, ic, assert
idtmod	abstol	expr, ic,modulus, offset
laplace_zp laplace_zd laplace_np laplace_nd	poles, abstol, zero	expr
last_crossing		expr, dir
limexp		expr
slew		expr, max_pos_slew_rate, max_neg_slew_rate
transition	time_tol	expr, td, rise_time, fall_time
zi_zp zi_zd zi_np zi_nd	zeros, poles, T , t0	expr, t

The Contribution Operator – '<+'

- *Accumulates potentials or flows to nodes, ports, and branches*
- **Order of contribution is not significant**

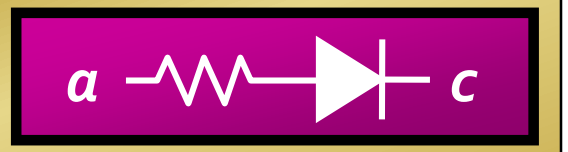
```
module rlc (a, b);
    electrical a, b;
    parameter R = 1 exclude 0;
    parameter C = 1;
    parameter L = 1 exclude 0;

    analog begin
        I(a,b) <+ V(a,b) / R;
        I(a,b) <+ C*ddt(V(a,b));
        I(a,b) <+ idt(V(a,b) / L;
    end
endmodule
```



The Contribution Operator – '<+'

- Supports implicit equations
  - Solves for *x* when *x* <+ *f(x)*

```
module diode (a, c);
    electrical a, c;
    parameter is = 1f from (0:inf);
    parameter rs = 0 from [0:inf);

    analog begin
        I(a,c) <+ is * ($limexp((V(a,c) – rs * I(a,c) ) / $vt) – 1);
    end
endmodule
```



Limiting Exponential (helps convergence)

I(a,c) on both sides makes eqn implicit

Example: Controlled Sources

```
module vcvs (p, n, ps, ns);
    electrical p, n, ps, ns;
    output p, n; input ps, ns;

    parameter gain = 1;

    analog
        V(p,n) <+ gain*V(ps,ns);
endmodule
```

```
module vccs (p, n, ps, ns);
    electrical p, n, ps, ns;
    output p, n; input ps, ns;

    parameter gain = 1;

    analog
        I(p,n) <+ gain*V(ps,ns);
endmodule
```

```
module ccvs (p, n, ps, ns);
    electrical p, n, ps, ns;
    output p, n; input ps, ns;

    parameter gain = 1;

    analog
        V(p,n) <+ gain*I(ps,ns);
endmodule
```

```
module cccs (p, n, ps, ns);
    electrical p, n, ps, ns;
    output p, n; input ps, ns;

    parameter gain = 1;

    analog
        I(p,n) <+ gain*I(ps,ns);
endmodule
```

Hysteretic Relay

```
module relay (pout, nout, pin, nin);
    voltage pout, nout, pin, nin;
    input pin, nin; output pout, nout;
    parameter real thresh = 0, hyst = 0;

    real offset;

    analog begin
        @(cross(V(pin,nin) – thresh – offset, +1))
            offset = –hyst;
        @(cross(V(pin,nin) – thresh – offset, –1))
            offset = hyst;

        if (V(pin,nin) – thresh – offset > 0)
            V(pout, nout) <+ 0;
    end
endmodule
```

Switch Branch

Event-Driven Modeling

- @ blocks
  - Blocks of code executed upon an event
- Event types

Name	Generates events …
cross()	At analog signal crossings
timer()	Periodically or at specific times
initial_step	At beginning of simulation
final_step	At end of simulation

- Time of the Last Zero Crossing; last_crossing()

Example: Record Zero Crossing Times

```
module zero_crossings (in);
    voltage in; input in;
    parameter integer dir=1 from [-1:1] exclude 0;
    integer fp; real last;

    analog begin
        @(initial_step)
            fp = $fopen( “zero-crossings”);
        last = $last_crossing(V(in), dir);
        @(cross(V(in), dir))
            $fstrobe( fp, “%0.10e”, last);
        @(final_step)
            $fclose(fp);
    end
endmodule
```

Record time of crossing

Example: Noisy Diode

```
module diode (a, c);
    electrical a, c;
    branch (a, c) diode, cap;
    parameter real is=1e–14, rs=0, tf=0, cjo=0, phi=0.7;
    parameter real kf=0, af=1, ef=1;

    analog begin
        I(diode) <+ is*($limexp(V(diode)/$vt) – 1);
        I(cap) <+ ddt(tf*I(diode) - 2*cjo*sqrt(phi * (phi * V(diode))));
        I(diode) <+ white_noise( 2 * `P_Q * I(diode) );
        I(diode) <+ flicker_noise( kf * pow(abs(I(diode)), af), ef);
    end
endmodule
```

4. Basic language features

Powerful syntax for declaring parameters with default and range:

```
parameter real r = 1000 from (0:inf);
```

Chained defaults:

```
parameter real l   = 1u     from (0:inf);
parameter real w   = 1u     from (0:inf);
parameter real rho = 1      from (0:inf);
parameter real r = rho*l/w  from (0:inf);
```

4. Basic language features

All behavior in the **analog** block

```
analog begin
    I(a,b) <+ V(a,b) / r;
end
```

Use **begin / end** for multi-line blocks – like you would use braces { } in C

*** Emacs and VIM have "Verilog-mode" plug-ins that can highlight keywords and keep your indentation correct.

4. Basic language features

The contribution operator:

```
analog begin
    I(a,b) <+ V(a,b) / r;
    I(a,b) <+ white_noise(4*`P_K
                *$temperature, "thermal");
end
```

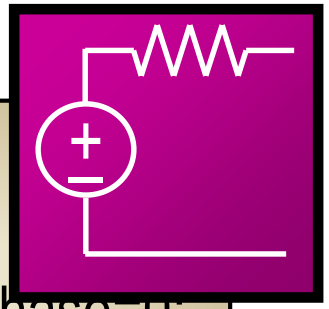
Not quite an assignment – all contributions to `I` are summed.

3. Simple examples

```
`include "disciplines.vams"

module simplecap(a, b);
   inout a, b;
   electrical a, b;
   parameter real c = 1000 from [0:inf);
   (* desc = "charge" *) real q;

   analog begin
      q = c * V(a,b);
      I(a,b) <+ ddt( q );
   end
endmodule
```

Always use
ddt(charge)
not C*ddt(V)

Example: Resistive RF Source



```
module port (p, m);
    voltage p, m;
    parameter real r=50, dc=0, mag=0, ampl=0, freq=0, phase=0;
    analog begin
        V(p,m) <+ 2*dc – r*I(p,m);
        V(p,m) <+ 2*ac_stim(mag);
        V(p,m) <+ white_noise(4*`P_K*r*$temperature);
        if (analysis("tran"))
            V(p,m) <+ 2*ampl*cos(2*`M_PI*freq*$abstime+phase);
        bound_step(0.1 / freq);
    end
endmodule
```