New enhancements in ADMS and Spectre CMI XML scripts

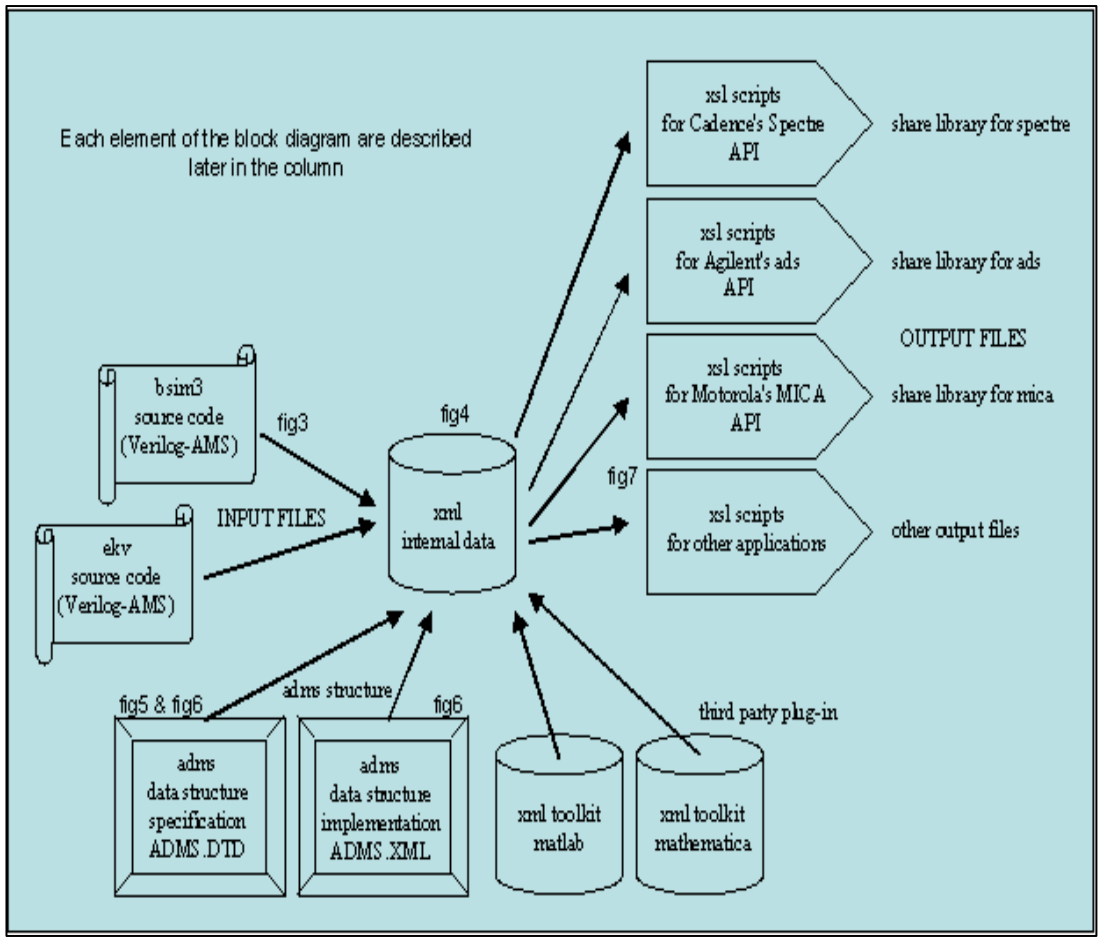
Sergey Sukharev

March 24, 2006, Workshop, Böblingen

Abstract

- *Automatic Device Model Synthesizer (ADMS) now provides the ability to create all required functionalities in Compiled-Model Interface (CMI) XML scripts for API specific simulators. This poster dedicated to approaches of implementation of most significant features in CMI XML scripts used by ADMS for Spectre. Approaches for implementation of new features were described and can be taken as template as a start of implementation other functionalities. Sometimes, there is no need to make changes in ADMS data tree, we change common properties of all simulators instead. ADMS is a translator which assembles using internal data tree Your unique code contained in XML scripts, – therefore intellectual property is safe. Close support of latest Verilog-A LRM is important. For example, support of voltage contribution and time integral operator allowed to translate more complex device models such as Hicum and PSP Verilog-A models, which become standard.*

Introduction into ADMS and XML technologies



ADMS is a code generator that converts electrical compact device models specified in high-level description language into ready-to-compile c code for the API of spice simulators. Based on transformations specified in xml language adms transforms Verilog-AMS code into other target language.

admsXml - interprets the admst intructions found in file myadmst.xml and apply the instructions to the contents of the verilog-ams file myverilogams.va.
admst path - how to navigate inside the adms internal data tree. admstpath gives the details of the yacc grammar used to build the admst path parser. The admst path is very similar to the xml xpath language. This is due to the limitations of the method applied to build the parser into adms. admst path have a lexical terminals.
adms internal data tree. After parsing an input file adms creates a tree - called adms internal data tree. The adms data tree is the internal representation of the parsed input.

Usage of ADMS tool

- A typical run of adms is shown below:

admsXml <vafile> -e <myinterface-file1>.xml -e <myinterface-file2>.xml -e ...

The language used to build the .xml files - called *admst* language – more described on mot-adms.sf.net.

- Why in adms uses admst and admstpath instead xslt and xpath?

  The reason is that the available xslt package was too slow at the time on investigations (mid 94) and some transforms (like admst:open) were not supported. Same thing for xpath. In the future plan to use xslt and xpath. (xpath will be the easiest thing to do since admstpath is a miniset of xpath.)

- What is the parsing/elaboration flow in ADMS?

  Here is the sequence:

<inputfile>

[parsed by admsPreprocessor]

<.inputfile.adms created>

[parsed by admsVeriloga]

<internal tree created that uses adms.xml data structure>

<.adms.implicit.xml is created>

[.adms.implicit.xml is parsed]

[at the same time all xml files are parsed]

<outputs created depending on contents of xml files>

- *Versions.* The latest version of ADMS is 2.2.0

where:

- 0 means changes connected with fix or improvement
- 2 means changes connected with XML scripts
- 2 means changes connected with changes in verilog-a parser

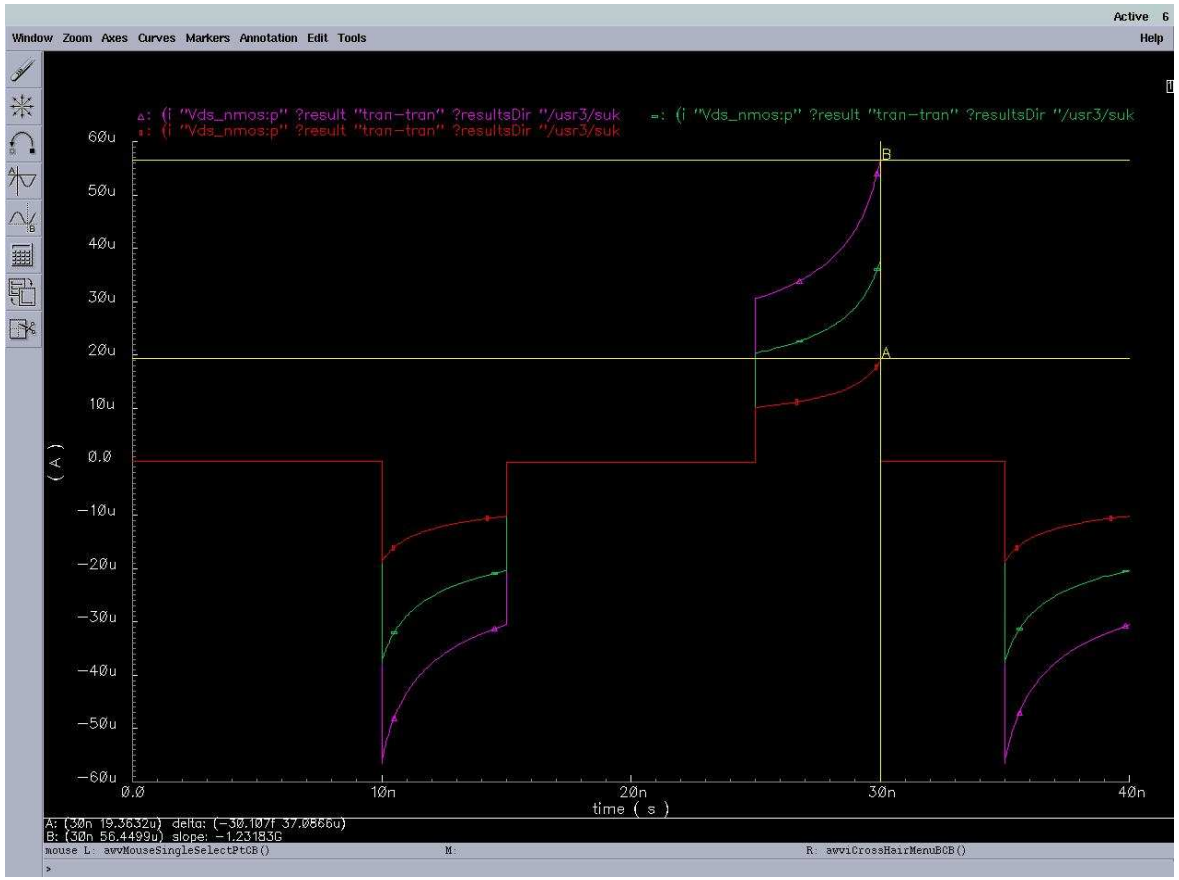
New implemented capabilities

In poster are presented new capabilities implemented in Spectre CMI XML scripts:

- *Multiplicity factor ($mfactor)*
- *Current and voltage contributions*
- *Time integral operator (idt)*

All capabilities were implemented according to Veriloga-A LRM-2.2 reference

Multiplicity factor in Spectre CMI XML scripts



Dependence on value m = [1..3] we can see different *Cbd*

$mfactor is the shunt multiplicity factor of the instance, that is, the number of identical devices that should be combined in parallel and modeled.

***Netlist with m-factor used with ADMS Spectre CMI bsim3v3, which illustrates dependence of value Cbd from m-factor

```
simulator lang=spectre
model mynmos admsbsim3v3
+ model parameters
…
…
Vds_nmos (d 0) vsource dc=0 \
type=pulse val0=0.0 val1=5 period=25n \
delay=10n rise=5n fall=5n width=10n

M0 (d 0 0 0) mynmos m='value' w=14u
l=0.35u ad=0.95u*(14u) \
as=0.95u*(14u) pd=0.95u*2+(14u) \
ps=0.95u*2+(14u) nrd=0 \
nrs=0 simulatorOptions
options …
tran tran stop=40n annotate=status
maxiters=5
```

Voltage and Current contributions

- Verilog-AMS HDL uses the *branch contribution operator* **<+** to describe analog behavior. This operator is only valid within an *analog block*. Branch contribution statements are statements which use the branch contribution operators to describe behavior in terms of a mathematical mapping of input signals to output signals.
- ***Branch.*** Each named branch is a separate physical branch in the network. There is only one branch created for an unnamed branch (All references to the unnamed branch are folded into one branch). For example, tree named branches are created as follows:

  branch (a,b) b1, b2, b3;

- **Probes.** A branch is a probe when it is only referenced on RHS of the contribution statement. There are flow and potential probes.

Equivalent circuit models for probe branches:

• ***Flow probe***

I(c, d) <+ I(b1);

Extra unknown	The flow through branch *b1*
Extra equation	V(b1) = 0
add/subtract I(b1) to KCL at nodes *a* and *b*	

• ***Potential probe***

V(c, d) <+ V(b1);

No extra equations/unknowns. Just use (V(a) - V(b))

FWI: A probe cannot be both flow and potential probe. i.e. You cannot reference both I(b1) and V(b1) on the RHS if b1 is only a probe.

Flow sources

- A branch becomes a source when it appears on the LHS of a contribution statement. A source branch must support probing of its flow and potential. (i.e. if a branch appears on the LHS, we can reference both it's flow and potential on the RHS).

- ***Flow source***

I(c, d) <+ (expression without reference to b1);

no extra equations/unknowns. Just add to KCL of nodes *a* and *b*

- ***Flow source with potential probe***

I(b1) <+ expression;
I(c, d) <+ V(b1);

no extra equations/unknowns. Just add to KCL of nodes *a* and *b* and use V(b1)=V(a)-V(b).

- ***Flow source with flow probe***

I(b1) <+ expression;
*I(c, d) <+ K * I(b1);*

Extra unknown	flow through *b1*
Extra equation	-I(b1) + (all contributions to I(b1))= 0
add/subtract I(b1) to KCL at nodes *a* and *b*	

- ***Flow source with both flow and potential probes:***

I(b1) <+ expression;
*I(c, d) <+ K * I(b1) + R * V(b1);*

Same as flow source with flow probe just use V(b1) = V(a)-V(b);

Extra unknown____	flow through *b1*
Extra equation	-I(b1) + (all contributions to I(b1))= 0
add/subtract I(b1) to KCL at nodes *a* and *b*	

Potential sources

- ***Potential source***

V(b1) <+ (expression without reference to b1);

Extra unknown____	flow through *b1*
Extra equation	-V(b1) + (all contributions to V(b1))= 0
add/subtract I(b1) to KCL at nodes *a* and *b*	

- ***Potential source with potential probe***

V(b1) <+ expression;
V(c, d) <+ V(b1);

Extra unknown____	flow through *b1*
Extra equation	-V(b1) + (all contributions to V(b1))= 0
add/subtract I(b1) to KCL at nodes *a* and *b* and use V(b1)=V(a)-V(b)	

- ***Potential source with flow probe***

V(b1) <+ expression;
*V(c, d) <+ K * I(b1);*

Extra unknown____	flow through *b1*
Extra equation	-V(b1) + (all contributions to V(b1))= 0
Add/subtract I(b1) to KCL at nodes *a* and *b*	
Use unknown for I(b1) in c, d KCL equations	

- ***Potential source with both flow and potential probes***

V(b1) <+ expression;
*V(c, d) <+ K * I(b1) + R * V(b1);*

Same as potential source with flow probe just use V(b1) = V(a)-V(b);

Extra unknown____	flow through *b1*
Extra equation	-V(b1) + (all contributions to V(b1))= 0
Add/subtract I(b1) to KCL at nodes *c* and *d*	
Use unknown for I(b1) in c, d KCL equations and use V(b1) = V(a)-V(b)	

Current Controlled - Current and Voltage Sources

```
//
// Current-controlled current source
//
//
// Downloaded from The Designer's Guide (www.designers-
guide.org).
// Post any questions on www.designers-guide.org/Forum.
// Taken from "The Designer's Guide to Verilog-AMS" by
Kundert & Zinke.
// Chapter 3, Listing 13.

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

module my_cccs (p, n, pc, nc);
    electrical p, n, pc, nc, pc_nc_probe;
    output pc, nc;
    input p, n;
    parameter real gain = 1.333;
    real Ipn;

analog
       begin

       Ipn = gain * I(pc,nc);
        I(p,n) <+ Ipn;

       // I(p,n)              <+ V(pc_nc_probe)*gain;
       // I(pc_nc_probe)  <+ V(pc,nc);
       // I(pc,nc)            <+ V(pc_nc_probe);

       end
endmodule
```

```
//
// Current-controlled voltage source
//
// Downloaded from The Designer's Guide (www.designers-
guide.org).
// Post any questions on www.designers-guide.org/Forum.
// Taken from "The Designer's Guide to Verilog-AMS" by
Kundert & Zinke.
// Chapter 3, Listing 13.

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

module my_ccvs (p, n, pc, nc);
    electrical p, n, pc, nc, p_n_flow, pc_nc_probe;
    output pc, nc;
    input p, n;
    parameter real gain = 1.333;


    analog
      begin

       V(p,n) <+ gain * I(pc,nc);

      //   I(p_n_flow) <+ V(pc_nc_probe)*gain;
      //   I(pc_nc_probe) <+ V(pc,nc);
      //   I(pc,nc) <+ V(pc_nc_probe);
      //   I(p_n_flow) <+ -V(p,n);
      //   I(p,n) <+ V(p_n_flow);

      end
endmodule
```

Time integral operator

- According to Verilog-A LRM 2.2, ‘idt’ is analog operator computing the time-integral of its argument.

idt(expr, IC)

When specified with IC, idt() returns the value of the IC in DC and IC analyses whenever assert is given and is non-zero. Without IC, it can only be used in a system with feedback which forces its argument to zero.

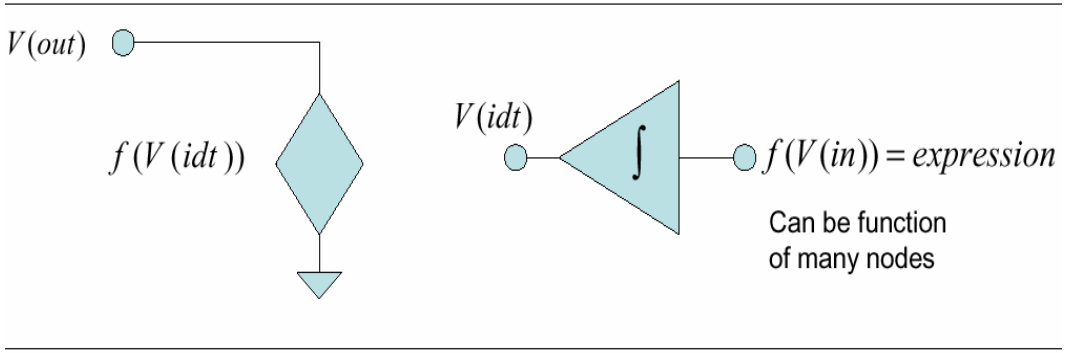
- Example with ‘idt’ contribution

V(out) <+ idt(V(in), IC, Reset)

Where: IC (double) - initial condition.

Reset (int) - When 1, the output is reset to IC.

Reset can be bias dependent. (it requires an evaluation).



Equations when Reset = 0:

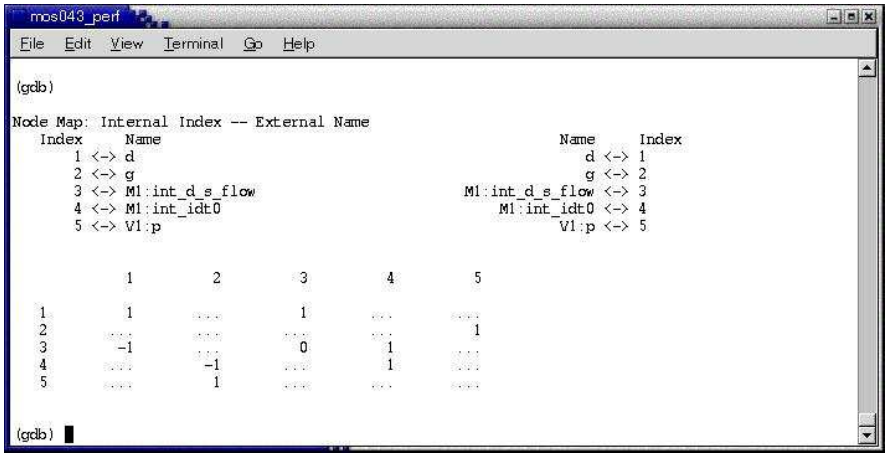
1: V(out) – f(V(idt)) = 0; Branch flow equation
2: ddt(V(idt)) + f(V(in)) = 0; Idt node equation

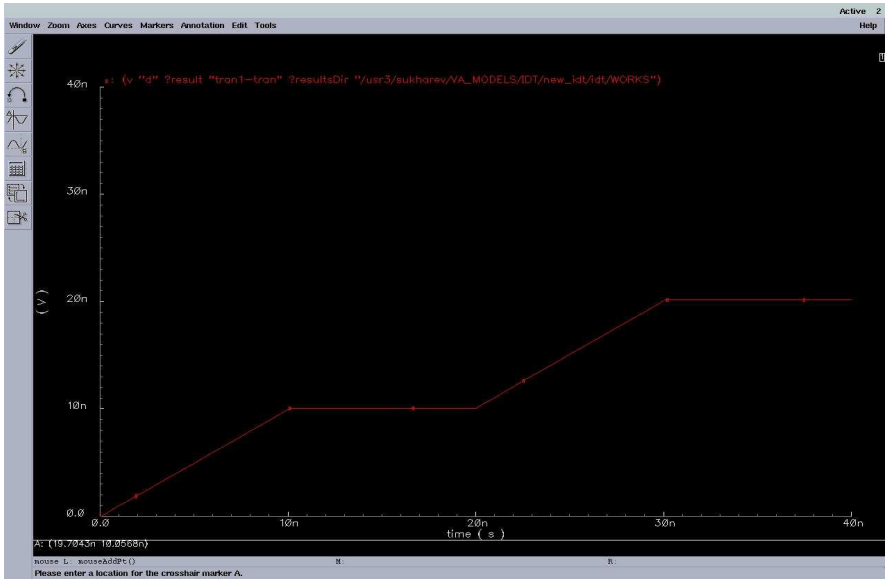
Idt( ) computation

```
// My Integrator
`include "discipline.h“
module my_mos(d,g,s,b);
inout d,g,s,b;
electrical d,g,s,b;
electrical d_s_flow, idt0;
parameter real IC=0.0;
analog     begin
//      V(d,s)  <+ idt(V(g,b),IC);
//      I(d_s_flow) <+ idt(V(g,b),IC);
I(idt0)     <+ -V(g,b);          // 1 during tran
I(idt0)     <+ V(idt0)-IC;       // 1 during DC
I(idt0)     <+ ddt(V(idt0));     //1 during tran this is zero
I(d_s_flow) <+ V(idt0);          // 2
I(d_s_flow) <+ -V(d,s);
I(d,s)      <+ V(d_s_flow);      // 2 and V contribution
end
endmodule
```

```
* Tran analysis
simulator lang=spectre
model test_idt my_mos
M1 (d g 0 0) test_idt
V1 (g 0) vsource dc=0.0 type=pulse val0=0 val1=1 delay=0 +rise=1e-10
+fall=1e-10 width=1e-08 period=2e-08
R1 (d 0) resistor r=1
tran1 tran stop=40n step=0.1n
save d V1:n
```

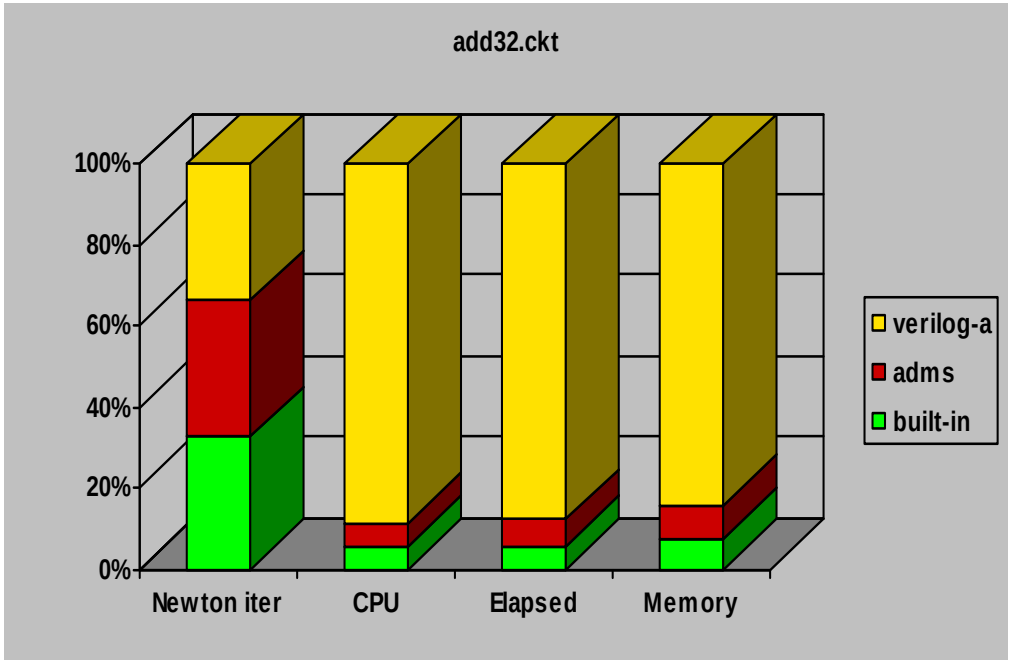
It is interesting to look at Jacobian of this netlist used ADMS Spectre CMI integrator





Performance

- Performance comparison between the hand-coded models, ADMS-generated models and the simulation using general Verilog-A flow was provided. BSIM3v3 was selected for comparison. Benchmarks were used as open source CircuitSim90 with add32.ckt, ram2k.ckt, mem_plus.ckt, sqrt.ckt, sram.ckt. So far as hand-coded model already optimized the source code getting by ADMS and Spectre CMI XML scripts was compiled with the same level of optimization.



There are five benchmarks with max quantity of transistors around 14000. In the performances were compared:

- Number of accepted tran steps;
- Number of Newton iterations;
- CPU time;
- Elapsed time;
- Virtual memory used;

Comparison between Verilog-A module, ADMS model and hand-coded (built-in) model was the following:

- Built-in model proved to be faster then ADMS model in maximum 10 %;
- And accordingly built-in faster then Verilog-a model in almost 10 times, but Verilog-a is a interpreter, not a model compiler yet.

Conclusion

- Our group adopted the ADMS model compiler technology to provide a link to the CMI used by Spectre and UltraSim. Cadence's early support for ADMS influenced the CMC's decision to adopt Verilog-A as a standard language for compact device models. We have worked closely with SONY, Freescale and University of Dresden (for HICUM model), to proliferate this technology. The ADMS tool offers an excellent modeling environment. It allows faster development of advanced models and faster implementation into commercial IC design tools. Furthermore, developers of new compact models now have access to the coherent and highly reliable modeling framework simplifying model evaluation procedures and verification tasks across different simulation platforms and operating systems.

The described approaches of implemented functionalities in Spectre CMI XML scripts makes us closer to translating and supporting of more complex models. We are going to continue efforts for investigations and implementations of new features according to Verilog-A LRM-2.2. The performance of ADMS devices is very important, too.

Thanks a lot for your attention