HICUM Status in Eldo/EldoRF

By
Mohamed Selim
Device Modeling
DSM/AMS

Overview

- Available HICUM versions n Eldo/EldoRF.
- Comments.
- Performance comparison.
- Conclusions.

Level 2

HICUM was first released in Eldo in October 1999.

Version	Implementation	Comment
V2.1	From the provided Fortran code.	Stable and reasonably fast.
V2.2	From provided Fortran code.	Same speed of v2.1, released in AMS2005.3 (Eldo v6.6_1.1)
V2.21	VerilogA modifications imposed over the Fortran code implemented v2.2	Same speed of v2.1, released in AMS2006.1 (Eldo v6.7_1.1)
V2.22	Fully implemented from the VerilogA code using the enhanced version of the VerilogA-2-C model implementation flow. (more optimized generated C code)	Compared to V2.21 it is rather slow. Released in AMS2007.1 (Eldo v6.9_1.1)

Level 0

Version	Implementation	Comment
L0- V1.0	Using the first version of the VerilogA-2-C model implementation flow.	Speed wise, poor compared to L2 v2.1 (hand optimized code). Released in AMS2005.1 (Eldo v6.5_1.1)
L0- V1.11 / V1.12	Using the enhanced version of the VerilogA-2-C model implementation flow. (more optimized generated C code)	Better speed. V1.12 released in AMS2006.2a (Eldo v6.8_2.1) while V1.11 released in AMS2006.2b (Eldo v6.8_3.1)

Comments

- Using ddx() function in VerilogA was not accepted by our implementation flow.
- ddx() is defined for VerilogAMS not VerilogA according to LRMs.
- Based on communications with Michael, we reverted back to V2.21 implementation for the code portions using ddx() as these parts in V2.21 were 100% equivalent to those in V2.22.
- NQS had to be added manually for L2 V2.22, not impossible but needs caution.
- The major issue seen so far for using the VerilogA as code standard is the versioning system.

Comments

- Each time a new version is out, whatever minor the changes are, we need to implement the new version from scratch. This adds code complexity (to the already complex, unreadable code) and redundancy for the code portions that are similar for the 2 versions.
- Suggestion: add version control over the VerilogA code so that each time a new version is added, it should be added over the older version.
- If a user needs to run the VerilogA code he can switch between different versions just be setting the appropriate value for the version parameter not switching VerilogA codes.

Performance Comparison

- Comparison done on a Linux 32 bits machine.
- Testcases used are all running transient simulations.
- 10 testcases were used for the comparison:

Test1	23 BJT Gilbert cell	Test6	6 BJT Latch
Test2	2 BJT amplifier	Test7	8 BJT Mixer
Test3	6 stage inverter, 12 BJTs	Test8	3 BJT Mixer
Test4	27 stage inverter, 27 BJTs	Test9	42 BJT design
Test5	L0: ECL inverter, 51 BJTs L2: gilbert cell, 23 BJTs	Test10	3 BJT design

Performance Comparison

- L2 V2.1 vs. L2 V2.22 to show the difference between implementation of hand optimized code and flow generated code.
  - Comparison is done for 2 libraries:
    - Default (no self heating or NQS , FLCOMP=2.1) This gave an average speed degradation of about 0.82x
    - Full (With FLSH=1, FLCOMP=2.1 and NQS enabled) This gave an average speed degradation of 0.82x

L2 V2.1 vs. V2.22 Full

	v2.1		v2.22			Gain = (v2.1-v2.22)/v2.1%		Speed up	
Circuit	Time(s)	devcall	Time(s)	devcall		time gain	devcall gain	v2.1 / v2.22	v2.22 / v2.1
Test1	30.84	463473	31.96	407997		-3.63 %	11.97 %	0.96x	1.04x
Test2	1.74	24992	2.42	30032		-39.08 %	-20.17 %	0.72x	1.39x
Test3	22.58	345168	28.01	342576		-24.05 %	0.75 %	0.81x	1.24x
Test4	57.22	1013148	75.22	1114749		-31.46 %	-10.03 %	0.76x	1.31x
Test5	11.45	170476	12.13	152099		-5.94 %	10.78 %	0.94x	1.06x
Test6	33.28	681600	56.29	934500		-69.14 %	-37.10 %	0.59x	1.69x
Test7	13.88	175704	23.98	248400		-72.77 %	-41.37 %	0.58x	1.73x
Test8	8.29	104643	8.60	89199		-3.74 %	14.76 %	0.96x	1.04x
Test9	91.00	1481718	92.14	1102614		-1.25 %	25.59 %	0.99x	1.01x
Test10	60.33	868635	67.65	841008		-12.13 %	3.18 %	0.89x	1.12x
Total	330.61	5329557	398.40	5263174		-20.50 %	1.25 %	0.83x	1.21x
Averages						-26.32 %	-4.16 %	0.82x	1.26x

Performance Comparison

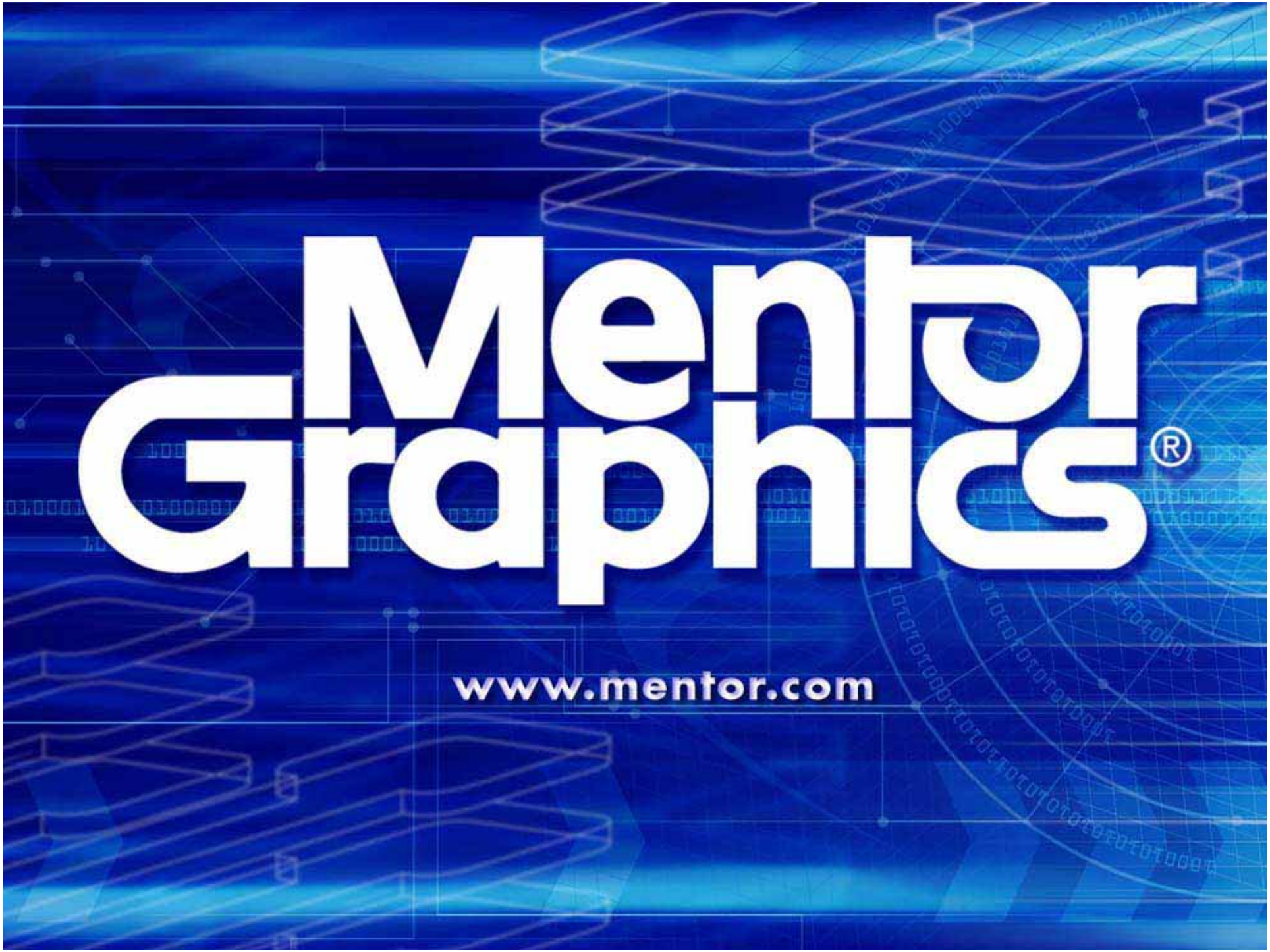
- L0 V1.0 vs. L0 V1.12 to show the performance enhancements in our VerilogA-2-C model implementation flow.
  - Comparison is done for 2 libraries:
    - Default (no self heating) V1.12 was faster by an average of 1.87x.
    - Full (with self heating) V1.12 was faster by an average of 1.39x.

L0 V1.0 vs. V1.12 Full

	v1.0		v1.12			Gain = (v1.0-v1.12)/v1.0%		Speed up
Circuit	Time(s)	devcall	Time(s)	devcall		time gain	devcall gain	v1.0 / v1.12
Test1	20.14	423235	10.40	239643		48.36 %	43.38 %	1.94x
Test2	13.21	219906	10.47	222156		20.74 %	-1.02 %	1.26x
Test3	3.42	14688	3.33	15004		2.63 %	-2.15 %	1.03x
Test4	9.27	5009	9.53	4922		-2.80 %	1.74 %	0.97x
Test5	25.09	358392	10.06	41077		59.90 %	88.54 %	2.49x
Test6	32.43	572953	31.85	721630		1.79 %	-25.95 %	1.02x
Test7	7.42	144672	5.35	125442		27.90 %	13.29 %	1.39x
Test8	4.56	68993	3.80	68985		16.67 %	0.01 %	1.20x
Test9	32.10	514447	23.62	429319		26.42 %	16.55 %	1.36x
Test10	43.49	700300	35.38	700263		18.65 %	0.01 %	1.23x
Total	191.13	3022595	143.79	2568441		24.77 %	15.03 %	1.33x
Averages						22.03 %	13.44 %	1.39x

Conclusions

- Implementation from VerilogA gives some speed degradation, v2.1 is faster than v2.22 (with flcomp=2.1) by about 25%.
- This might be acceptable due to newer machines with higher processing power.
- Still the generated code has room for optimization from the implementation flow point of view.
- Due to enhancements in the VerilogA-2-C device models implementation flow in Eldo we were able to get better code that runs faster from 35% to 90%.
- We need to address the versioning system to facilitate the implementation process and set some standards to be followed by developers.


Mentor Graphics®
www.mentor.com