Verilog-A module for HfOx RRAM Resistive Memory Elements

# Introduction

This document describes practical usage and potential issues for the verilog-A module that was developed at imec for RRAM Resistive Memory Elements (RMEs) with behavior similar to that of typical HfOx cells. This document is targeted at people who have to use or modify the model.

Contact Stefan Cosemans ([stefan.cosemans@imec.be](mailto:stefan.cosemans@imec.be)) for more information or in case problems are encountered.

Last updated on December 22nd, 2011

# Running the examples

* Running the simulation
  + “cd” to the ./netlist/ directory.
  + Make sure you have sourced your simulator environment
    - Doublecheck whether the “logs” subdirectory exists**!!!**
    - Otherwise, you will get an error:
      * Hspice and hspice64: \*\*error\*\*: invalid memory reference
      * Hspice –hpp: \_\*\_ internal S-fault \_\*\_
  + “hspice setDC.sp”, “hspice64 setDC.sp”, “hspice setDC.sp –hpp”
    - Change the simulation statements at the bottom of the netlist according to the way you want to visualize the results:
      * Standard wavebrowser: use “.option POST=2”. Disable “CSDF”
      * Matlab scripts: use “.option CSDF” and disable “\*.option POST=2”
  + “spectre setDC.sp”, “aps setDC.sp”, …
    - Change the simulation statements at the bottom of the netlist:
      * In the simulation section, disable all spice commands except for the “.tran step=x stop=y” line. Enable the spectre section.
  + Examples [see PTW Q4 2011 presentation]:
    - setDC.sp : slow transient simulation mimicking DC set behavior
      * Note: we use a slow transient simulation rather than a real DC simulation as the model does not explicitly support DC simulations (it wouldn’t make much sense with the time constants etc.).
      * Contains both simulations from the PTW presentation. Select the simulation at the top of the netlist file
    - resetDC.sp
      * Contains both simulations from the PTW presentation.
    - transientSimulation.sp
    - transistorSimulation.sp
    - matrix/(generated/)matrixSimulation.sp : see below
  + Notice: spectre (and aps) seem to work best with the analytical QM sconduction model. For hspice, we had to replace the QM conduction model equations with a lookup-table. This lookup table is written from a matlab script.
* Plotting graphs
  + Using normal wavebrowsers (cscope)
    - Hspice (e.g. cscope) : make sure to enable “.option POST=2”
    - Spectre: it is more efficient to disable the “rawfmt=psfascii” line, but psfascii should work with “viva”
  + Using matlab scripts [advised]
    - Install the signalStorageToolbox, which should have been provided as a separate package. If you didn’t receive it, contact Stefan Cosemans.
    - Make sure to perform the simulation with the following options:
      * Hspice: “.option CSDF” enabled, “.option POST=2” disabled
      * Spectre: “save=all” and “rawfmt=psfascii” enabled
      * Notice plot scripts have been written for hspice. Can be converted to spectre (or both) on demand.
    - Start matlab
    - “sourceSignalStorage”
    - “cd” to ./netlist/visualize/”
    - “setPlots” [ or “close all; sim=setPlots(); ]
      * Use sim.getAllNames() to see which signals have been saved. Refer to the signalStorageToolbox documentation for more information
    - Similarly “setPlots”, “resetPlots”, “transientPlots”, “transistorPlots”
    - Notice: “WARNING: the requested signal ‘xyz’ does not exist” are ok.
* Running the matrix example:
  + The matrix example is located in ./netlist/matrix
    - Optional: ./netlist/matrix/matrixSimulation.m2s is the “mat2spice” description of the netlist, which allows to write matrices of different sizes etc. Use “createNetlist()” to generate an actual netlist. The netlist is written to “./netlist/matrix/generated/matrixSimulation.sp”. This requires that the mat2spice toolbox is available and in the path. Ask Stefan Cosemans for more information.
    - A generated 8x8 matrix can be found in ./netlist/matrix/generated/matrixSimulation.sp .
      * Run “hspice matrixSimulation.sp [-hpp]” or “spectre matrixSimulation.sp”.
      * Notice that we have encountered issues when running plain hspice (without –hpp) on 64 bits machines (see the issues section below).
    - In matlab, execute “plotMatrixSimulation” in ./netlist/matrix/ to view the result of the simulation
    - Notice that the applied waveforms are a bit simplified: one WL is activated and stays open for all accesses, only the BL signals change during the simulation. This is just for convenience, not to hide simulation issues.

# Using the RRAM RME model

There are two options: use the analytical QM conduction model (best for spectre and aps), or use lookup tables for the I-V curves instead. Both options allow simulation of set and reset operations. Hspice requires the lookup tables, spectre works best with the analytical equations.

* Analytical model
  + Include “./verilog/RME\_QMMODEL\_ANALYTICAL/RME\_QMMODEL.va” with a “***.hdl “includefile.va”***” statement in the netlist.
  + Instantiate the RME with
    - Xrme top\_terminal bottom\_terminal log10\_omega\_y\_XY RME\_QMMODEL [parameters]
    - In the list of parameters, specify “initial\_omega\_x” and “initial\_V0”
    - Notice: names in blue are node names. Make sure you don’t connect different “log10\_omega\_y\_XY” nodes together! [When we find out how to save waveforms for signals that are internal to a verilog-A module for all simulators (“hspice –hpp” in particular), this node will be removed from the node list.
* Lookup tables
  + Write the verilog-A module and lookup table for the desired values of (V0,omega\_x), over the required range of omega\_y and Vrme.
    - Matlab code to write the verilog-A file and the lookup table file can be found in “./writeRME\_QMMODELs/”
    - “RME\_QMMODEL.template.va” is the main description of the verilog-A model for the RME. Changes should be applied to this file, not to the generated files.
    - Edit write\_RME\_QMMODELs.m according to the RME parameter values you need
    - Exectue write\_RME\_QMMODELS(). Make sure to perform a visual check on the generated graphs. You might have to enable “generatePlots=1” in this matlab file.
    - Notice: two example modules are provided in the distribution in ./verilog/RME\_QMMODEL\_TABLES/
    - In case you have no access to matlab, You can ask specific verilog-A modules from Stefan Cosemans. An individual matlab license is about 2K$.
  + For a use case of the lookup-table models, see e.g. the setDC.sp examples that are provided in the distribution. The essential change compared to the analytical model is
    - When using the analytical model, you specify omega\_x and V0 as instance parameters of the verilog-A instance, and all RMEs use the same verilog-A module
    - When using the lookup-table based model, you create a verilog-A module for each specific (omega\_x, V0) value, and the RME instance refers to that specific module, e.g. RME\_QMMODEL\_0000\_700THz\_m0050\_000uV for a RME with V0=-50mV and omega\_x=700e12 Hz = 7e14 Hz

# Changing the verilog-A model

In case you would have to change the verilog-A model yourself, it is best to

* change “./writeRME\_QMMODELS/RME\_QMMODEL.template.va”
* Generate both analytical and lookup-table based verilog-A files using “write\_RME\_QMMODELs.m”
* Run ./installModels.sh to copy the generated files to the ./verilog/RME\_QMMODEL\_XYZ directories
* Other verilog code (helperFunctions.v, analyticalModel.v and PERIPHERY.va) can be changed directly in ./verilog/

# Known issues

## Always use accurate simulation settings

* Spectre: use e.g. “myTran tran stop=1e-6 errpreset=conservative”
* Hspice: use e.g. “.option accurate”
* Simulation time doesn’t seem to be heavily affected, but the results are significantly more correct…

## Issues with compilation of verilog-A code

* If compilation of verilog code fails, this might be due to a problem with 32/64 bit machines or an incorrect system setup of the machine.
* **Hspice**: Running 32 bit hspice on 64 bit computer or the other way around:
  + \*pvaI\* system & gcc return code is 512. \*\* error \*\* call to epvaHDLcgen failed […]
  + On some installations, the command “hspice” refers to the 32 bit binary. On a 64 bit machine, compilation of the verilog code can fail. Use “hspice64” instead.
* **Spectre**: set environment variable “CDS\_AUTO\_64BIT” to ”ALL”, e.g. by adding
  + *export CDS\_AUTO\_64BIT=”ALL”* to ~/.bashrc
* It is possible to compile the verilog-A code on a 32 bit machine first (by starting a simulation), and then to use this pre-compiled verilog code on the 32 bit version of hspice on a 64 bit machine.

## Issues with hspice

* Hspice64: “\*\* error \*\* invalid memory reference” (command exited with non-zero status 11)
  + This error seems to occur with the 64 bit version of hspice when there are more than ~6x6=36 RRAM RME instances. It occurs independently of whether the lookup table is used or not: it happens even when the current equation is replaced with “I(t1,t2) <+ 0;” [keeping all the other code identical]. This problem does not seem to occur with “hspice64 –hpp”.
* Convergence issues when using analytical QM model
  + Hspice has significant convergence issues when the QM IV model is used directly, although it does calculate the correct current values when the RME is connected directly to a voltage source or a small resistance, instead of through a larger resistance.
  + Therefore, we switched to lookup tables for hspice simulations.
* Make sure to enable the “.option accurate” line, unless you know what you are doing. It doesn’t seem to affect simulation speed much anyway.

## Issues with hspice – hpp

* Unlike for spectre and plain “hspice”, we haven’t found a way to save the waveforms for the internal electrical nodes when using “hspice – hpp” due to a lack of documentation. For the time being we added log10\_omega\_y as an additional output node for the RME:
  + Xrme top\_electrode bottom\_electrode log10\_omega\_y **RME\_MODEL**
  + The other signals (omega\_y\_target, time constants) are not saved and hence also not plotted in the visualization scripts
* It appears as if “hspice –hpp” is creating a separate lookup table for each RME instance. This is not the case with spectre, nor with plain hspice.
  + Creating instances takes quite a while
  + It is fairly easy to exceed available memory… (16x16 cells of the example 1.9MByte tables consume 2.6GBytes RAM)

## Issues with the analytical QM conduction model (for I-V curves)

* “hspice“ has significant convergence issues with the analytical conduction model equations. Use the lookup table models instead… Results with “hspice –hpp” are even worse…

## Issues with lookup-table implementation for I-V curves

* Spectre works significantly slower with the lookup tables than with the analytical model
* When too few points are used when writing the lookup table, and linear approximation is used in the verilog-A $table\_model( … ) command, you can observe weird (but smooth) wavy behavior in the currents. Those are a direct consequence of the lookup table implementation, not simulation artifacts. The effect can be reduced by
  + Using higher-order interpolation. Add “3,3” as last argument to the $table\_lookup( … ) function. This makes the simulations significantly slower.
  + Use more points in the lookup table. Simulation speed shouldn’t be affected much, but lookup table file size increases a bit, but as long as we only need 2 input parameters (Vrme and omega\_y), this should not be a problem.

## Various issues

* Hspice –hpp: “*-\*- internal S-fault -\*-*“ :Error occurs when ./logs/ subdirectory does not exist and the RME tries to create the log file in that directory
* You can’t have a separate log file for each RME: spice or spectre (don’t remember) only allows 1K open files. If you exceed this limit, no output signal database is written. Therefore, all logs are written to the same log file ./logs/all\_RMEs.log