Coho Circuit Modeling (Coho-Model) Tool Manual

Chao Yan

November 25, 2014 (Version 1.0)

Contents

Chapter 1

Introduction

A fundamental step of circuit simulation is to model circuit devices accurately and efficiently. Usually, circuit modeling is a complicate and tedious work for researchers and simulator developers. COHO *Circuit Modeling* (CCM) is a tool for extracting mathematical models from circuit layout automatically. It has features:

- CCM uses fabrication raw data, instead if SPICE-like models. SPICE-like models are abstraction of physical devices. They are not accurate enough for analog designs, especially for deep sub-micron process, *eg* 14nm. Due to the inaccuracy of these models, the simulation results usually do not match with real circuits. Our approach uses data from factory, thus could be arbitrary accurate.
- CCM provide mathematical models for both simulators and formal verification tools. For simulators, it provide ODE models; and for formal verification, it provides *Linear Differential Inclusion (LDI)* models. This enables to compare the results of simulation and reachability analysis.
- CCM is indepdent of the circuit simulator or reachability analysis tools (*e.g.* CRA). The interface is simple mathematical models *i.e.* ODE for simulators and LDI models for reachability analysis tools.
- CCM is very flexiable: easy to add or change device models or circuit libraires. It is also easy to be extended to support cutomized features.

1.1 Installation

CCM is open-sourced. Users can download from *github* by:

```
git clone https://github.com/dreamable/ccm.git ccm
```

It is purely MATLAB programs, thus support all operation systems. It can be installed by:

```
cd ccm
sh install.sh
```

1.2 Simple Usage

CCM is a MATLAB package. To use it, please first start MATLAB, then run your MATLAB codes by:

```
cra_open
%user_code
cra_close
```

1.3 Examples

Examples are available under the following directories in the source code:

- example
- MSpice/test
- libs/coho/test

1.4 Organization

2 describes the usage of CCM and its mathematical model interface for simulators reachability analysis tools. 3 describes the interpolation methods and linearization methods. 4 describes the default circuit libraries and how to add users' own libraries. 5 describes the default device models and how to add new models.

1.5 Learn More

There are documents in the *doc* directories. There are also papers published which are good sources to understand the higher level ideas.

Table 1.1: Publications

Publication	**Note**
[1]	This is Chao Yan's PhD thesis. It is a comprehensive document with most details. Circuit modeling are covered in chapter 3.

To understand implementation details, please use the MATLAB help files by

```
help funcName
```

Chapter 2

MSpice

MSpice is the interface of CCM provided to users. It is based on two classes *circuit* and *testbench*.

Here is an example of how to simulate an inverter circuit

```
c = inverter('inv',1e-5);
p = vpulse('input',[0,vnn],[2,8,2,8]*1e-10,1e-10);
s = testbench(c,{c.i},{p});
[t,v] = s.simulate([0,1e-8]);
```

First, we create an *inverter* circuit, *'inv'* is the name of the circuit, and the size is 1e-5. Second, we proivde an input waveform by 'vpulse', where the rising/falling time is 2e-10, and high/low time is 8e-10. Then, we can create a testbench for the inverter circuit, with input connect to the vpulse. Finally, we can get the simulation by calling the *simulate* function.

2.1 Interface

The *circuit* class is the base class of all circuits. It provides the following functions

- Properties:
  - name: circuit name
  - nodeNum: number of circuit nodes
  - ports: visible circuit nodes
- Simulation interface:
  - i = I(v): node current
  - c = C(v): node capacitance
  - dv = dV(v): ODE models

  - v = V(t): node voltage, for voltage sources only

- Formal verification interface:
  - ldi = I_ldi(region): LDI models of node currents.
  - ldi = dV_ldi(region): LDI models of circuit dynamics.
- Small signal interface:
  - j = Jac(v): Jacobian matrix of circuit dynamics
- Utilities:
  - ifc_simu: support simulation interface
  - is_vsrc: is voltage source or not
  - ifc_verify: support verification interface
  - ifc_ssan: support small signal analysis interface
  - print_circuit_tree: show circuit hierarchy tree
  - print_nodes: show circuit nodes with descriptions

The *testbench* class is a wrapper of *circuit*, driving the input signals for constructing a complete simulation/verification environment. For each input signal, user should provide a voltage source to drive it. The *testbench* class provides:

- $[t, v]$ = simulate(tspan,v0,opt): default simulator
- dv = dV(v): full dimensional ODE models for simulator.
- ldi = dV_ldi(region): full dimensional LDI models for verification.

2.2 How to create new circuits

Circuit is a connected set of sub-circuits. To create a new circuit, a subclass of the *circuit* class should be defined. As illustrated in the *NAND* example below, there are three steps: 1. define the circuit nodes; 2. define the sub-circuits; and 3. define the connections.

```
classdef NAND < circuit
  properties (GetAccess='public', SetAccess='private');
    i1,i2; o; % input/output
  end
  methods
    function this = NAND (name,wid,rlen)
      this = this@circuit(name);  % super-class constructor
```

```
      % define circuit nodes
      this.i1  = this.add_port(node('i1'));
      this.i2  = this.add_port(node('i2'));
      this.o   = this.add_port(node('o'));
      x        = this.add_port(node('x'));

      % define sub-circuits
      n1 = nmos('n1',wid(1),0,rlen); this.add_element(n1);
      n2 = nmos('n2',wid(1),1,rlen); this.add_element(n2);
      p1 = pmos('p1',wid(2),1,rlen); this.add_element(p1);
      p2 = pmos('p2',wid(2),1,rlen); this.add_element(p2);

      % define connections
      this.connect(this.i1,n1.g,p1.g);
      this.connect(this.i2,n2.g,p2.g);
      this.connect(this.o,n1.d,p1.d,p2.d);
      this.connect(x,n1.s,n2.d,c.x);

      this.finalize; % complete
    end
  end
end
```

Chapter 3

Interpolation and Linearization

CCM provides a set of interpolation methods to generate ODE models for simulations, as well as linearization methods for generate LDI models for verification.

3.1 Interpolation

We provide three interpolation methods:

- lookup: find the nearest grid points and use its value
- linear: find all neighbors and use bilinear interpolation
- coswin: find all neighbors and use 'cosine window' interpolation.

The default interpolation method is 'coswin'. The default value can be updated by

```
ccm_cfg('set','interpMethod','coswin/linear/lookup');
```

The interpolation methods are independent of device model types, *i.e.* are the same for both 'simu' and 'quad' models.

The continuity of the interpolated function is The coswin method can provide continuous functions with arbitrary orders. By default, the interpolated functions are of C^5.

	simu	**quad**
lookup	not continuous	not continuous
linear	C^0	C^0
coswin	C^n	C^n

3.2 Linearization

Linearization methods are different for different models. For 'simu' type models, CCM provides

- 'lls': linear least square method, the default method.
- 'mm' find the minimum and maximum value. The error term is usually much larger.

For 'quad' type models, CCM provides

- 'pt': compute linear differential model for quadratic model with optimal L2 norm error
- 'lip': compute an approximate result based on Lipschitz constant. The is usually more efficient, while the error term is slightly larger. The default method.

Similarly, the default method can be updated by

```
ccm_cfg('set','lftSimuMethod','lls/mm');
ccm_cfg('set','lftQuadMethod','pt/lip');
```

3.3 Jacobian

We provide both analytical and numerical methods for computing Jacobian matrix.

The analytical methods uses the 'coswin' interpolation methods for evaluating the function value. The 'lookup' and 'linear' interpolation are not C^1, thus can not be used. And it is not recommended to use 'simu' models with analytical methods.

The numerical method can use all three interpolation methods. However, the result is usually bad for the 'lookup' method. It's OK most time for the 'linear' method. For the 'coswin' method, it works well for the 'quad' models, but not 'simu' models.

Similarly, the default method can be updated by

```
ccm_cfg('set','interpJacMethod','ana/num');
ccm_cfg('set','interpJacNumMethod','linear/coswin/lookup');
```

Chapter 4

Circuit Libraries

4.1 Coho Libraries

The default COHO libraries provides basic MOS circuits and RC circuits. For details, please check *./libs/coho*. The library supports all interfaces of the *circuit* class. It also support independent current sources, and voltage sources including: independent voltage source, voltage pulse, sine/cosine voltage waveform, and the Brockett's annulus for both simulation and verification.

4.2 How to add new libraries

New libraries should provides functionalities for all leaf-circuits, which have no sub-circuits. For non-leaf circuits, the *circuit* abstract class will provide all functions automatically. The leaf circuit subclass must override the following functions:

- I: (if ifc_simu & is_vsrc)
- C: (if ifc_simu & is_vsrc)
- V: (if ifc_simu & is_vsrc)
- V_ldi: (if ifc_verify)
- dIdV: (if ifc_ssan)
- Static.I_objs (for performance)
- Static.C_objs (for performance)
- vectorize_subtype_id (only if class may have different configurations)

To add the new library, the library path should be set by

```
ccm_cfg('set','libRoot','<libpath>');
```

Chapter 5

Device Models

5.1 Default models

CCM provide PTM process models by defaults. It provides two types of models: 'simu' and 'quad' models. The 'simu' type of models are raw data from factory process, which are essentially a huge table of sampled data. The 'quad' models are typically smaller. It use quadratic polynomial functions to approximate the raw data. It provide models for leaf circuits 'nmos', 'pmos', 'inverter', 'nmos with source connected to groud', and 'pmos with source connected to power'.

5.2 How to add new models

To add new device models, user need to add the path by

```
ccm_cfg('set','matRoot','<matpath>');
```

By default, the mat file is located by

```
<matRoot>/uppper(fab)/lower(process)/lower(type)/lower(circuit).mat
```

e.g './mat/PTM/180nm/simu/nmos.mat'. This can be changed by

```
ccm_cfg('set','matFileFunc','<func>');
```

For each device model, the 'mat' file must have

- GRID: sampling grid information
  - v0: initial value for grid '0', dx1 vector
  - nv: number of grid points, dx1 vector
  - dv: unit length of grid, dx1 vector
- data: model data for each grid point, n-dimentional array
- err: model error bounds for each grid, same size with 'data', could be [] if no error, or scalar if same for all points.

- SIZE: device size
  - len: device length
  - wid: device width
- META: additional information
  - type: 'simu' or 'quad'
  - *lib: library name (* for optional)
  - *name: device name
  - *fab: fab, e.g. TSMC, INTEL, etc
  - *process: e.g. 180nm
  - *gnd/vdd: voltage of gnd/vdd
  - *nodes: name of device nodes
  - *desc: description of this model

Functions in './Utils/model' are helpful for generating such device models.

Chapter 6

Advanced configuration

CCM can be configured by

```
ccm_cfg('set',k1,v1,k2,v2,...);
```

The following options can be configured

- matRoot: root directory for the model files.
  - default: CCM_HOME/mat
- matFileFunc: the function for finding mat files given a device
  - default: ¡fab¿/¡process¿/¡type¿/¡name¿.mat
- fab: fabrication information
  - default: 'PTM'
- process: process information
  - default: '180nm'
- type: model types
  - values: 'simu','quad'
  - default: 'simu'
- libRoot: root directory for the libraries.
  - default: CCM_HOME/libs/coho
  - NOTE: this must be set before 'ccm_open'.
- interpMethod: default interpolation methods for current function
  - values: 'coswin','linear','lookup'
  - default: 'coswin'

- interpJacMethod: default interpolation methods for Jacobian matrix
  - values: 'ana'(analytical),'num'(numerical)
  - default: 'ana'
- interpJacNumMethod: default interpolation methods for numerical Jacobian matrix
  - values: 'linear','coswin','lookup',
  - default: 'linear'
  - NOTE: The result of 'lookup' method is bad, the 'coswin' method for 'simu' models is bad.
- lftSimuMethod: default linearization methods for Simu models
  - values: 'lls','mm','ls','lp'.
  - default: 'lls'
  - NOTE: 'ls' and 'lp' are very expensive
- lftQuadMethod: default linearization methods for Quad models
  - values: 'lip','pt'
  - default: 'lip'

Please check ' help ccm_cfg.m' for details.

Chapter 7

Examples

Bibliography

[1] Chao Yan. *Reachability Analysis Based Circuit-Level Formal Verification.* PhD thesis, The University of British Columbia, 2011. 6