Master Thesis

OpenModelica for Analog IC Design

David Malo Cid
Linköping University
Linköping
Sweden

Examiner:

Dr. Mohsen Torabzadeh-Tari
Department of Computer Science

Dr. Jacob Wikner
Department of Electrical Engineering

Linköping University, Linköping

August 23, 2011

LITH-IDA/ERASMUS-A---11/001--SE

Table of Contents

Table of Figures

Table of Equations

Acknowledgments

I would like to thank Dr. Mohsen Torabzadeh-Tari for his help and advice as well as Dr. Jacob Wikner and PELAB colleagues, especially Mr. Syed Adeel Asghar. Also I would like to thank my parent and friends for supporting me during all these months.

Abstract

Modelica is a language supported by Modelica Association. It is a non-proprietary, object-oriented, equation based language to conveniently model complex physical.

OpenModelica is an open source environment for the Modelica language supported by Open Source Modelica Consortium (OSMC). [1]

The aim of this thesis is the development of a library of models and components for design and simulation of analog integrated circuits and an OMNotebook-based tutorial for an academic purpose as a design tool for an electronics lab.

For implementation of this library we got inspired from SPICELib library [2], but we have tried to make it more user-friendly without losing model complexity.
The first difference is how analyses are developed. In our library DC, AC and transient variables are calculated for each simulation while in SPICELib you have to instantiate an analysis model and execute it.
Also we have included length and width as parameters in most of the components, what is not included in SPICELib.
Another difference is the way voltage and current sources are edited which we have tried to do it easier than in SPICELib.

However we have decided to keep the model used for NMOS and PMOs transistors (but with analysis features described before) since it has enough complexity for our purpose that is, by now, that our library can be used as integrated circuits simulator for ICs courses lab.

To add functionality similar to other simulation tools, we have modified a python script that lets you make a parametric sweep, something that was not available in SPICELib.

We have used a script already implemented in [3], which used to simulate a model several times after changing one of the parameters of that model.

Using this script as a reference we have adapted it to do those simulations, change the parameter we want after each simulation and finally, plot one of the variables of the model as a function of the parameter that was changed.

Outline

We have divided this paper in seven chapters:

Chapter 1 - Introduction: it is given a brief introduction about concepts like Modelica or OpenModelica and it is also explained the motivation and results of this thesis.

Chapter 2 - Library description: in this chapter we describe the library structure, showing the content of each sub-package.

Chapter 3 - Library models: library models are described more in detail explaining their parameters and equations.

Chapter 4 - Getting started: explains how to use our library to create, simulate and plot integrated circuits.

Chapter 5 - Test cases: some circuit examples are presented and results of their simulations are showed as a test for our library.

Chapter 6 - Complementary modules: in this chapter it is explained two complementary modules made apart from library. These are aPython script and an OMNotebook tutorial.

Chapter 7 - Conclusions and future works: we conclude with this chapter where we include some conclusion and future works which could be made realted to our library.

1. Introduction

In this chapter we will introduce the language (Modelica) and the environment (OpenModelica) used for the creation of the library as well as some concepts about analog integrated circuits.

1.1 Modelica

As it is told in Martin Otter and Hilding Elmqvist article, Modelica is a freely available, object-oriented, equation-based language for modeling of large and complex physical systems. It is suited for mechanical, electrical, hydraulic and control subsystems modeling, [1].

Modelica models are defined by differential, algebraic or discrete equations which are solved by a Modelica tool, so it not needed to do it manually. Specialized algorithms can be utilized to efficiently handle this equation solve in large models having more than a hundred thousand equation, [1].

Modelica supports high-level models for composition and detailed modeling of library component. Library contains models for standard components allowing the user to create more complex models by positioning, connecting and setting parameters of standard components as well as creating completely new ones.

1.2 OpenModelica

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for both industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC), [5].

The main goals of this project are:

- Development and long-term support of an open source industrial-strength Modelica modeling and simulation environment for industrial and academic usage.
- Creation of a complete reference implementation of Modelica in an extended version of Modelica itself. Promoting a model-centric approach to give the modeler increased control, including modeling of both hardware- and software components in complex systems.
- Improved implementation techniques, e.g. to enhance the performance of compiled Modelica code by generating code for parallel hardware.
- Research in areas such as language design, control system design, debugging support, embedded system modeling, new symbolic and numerical algorithms, etc.

OSMC is a non-profit, non-governmental organization with the aim of developing and promoting the development and usage of the OpenModelica open source implementation of the Modelica language and OpenModelica associated open-source tools and libraries, collectively named the OpenModelica Environment, [5].

1.3 Analog Integrated Circuits

"Integrated circuits (ICs) consist of miniaturized electronic components built into an electrical network on a monolithic semiconductor substrate by photolithography", [6].

Because of this miniaturization it is becoming more and more necessary a good process of design. This is one of the main reasons behind the desire of constructing an easy self-mantained library.

Since this is a basic implementation of an IC design library, we have included the most typical components.

These components are:

- Resistor
- Capacitor
- Inductor
- NMOS transistor
- PMOS transistor

We have design them so that the only parameters that can be directly changed are length and width. Obviously there are many other parameters, especially in transistors, which could be altered in the library before creating models.

Other components have also been created to make the library completely autonomous. These components are:

- Ground
- Current source
- Voltage source

1.4 Motivation

The motivation of this thesis is to test the possibility of using OpenModelica as a simulation tool in a lab environment focused on analog integrated circuits.
This is the reason behind the development of our library apart from that existing libraries where more PCB circuits oriented making them not as convenient to integrated circuits design as an integrated circuits oriented library.

Obviously, considering the aim of our thesis, we do not include all possible components nor all possible analyses which you could find in an advanced simulation tool, just most commonly used ones.

1.5 Results

The result of this project is an integrated circuits oriented library which takes advantage of all the usefull features of Modelica language and OpenModelica environment to create a user-friendly simulation tool for electronics lab environment.

2. Library Description

During the implementation of the library we have tried to follow the scheme and naming conventions already used in Modelica Standard Library. Therefore, we have divided our library into six sub-packages:

- Components
- Constants
- Functions
- Interfaces
- Sources
- Examples

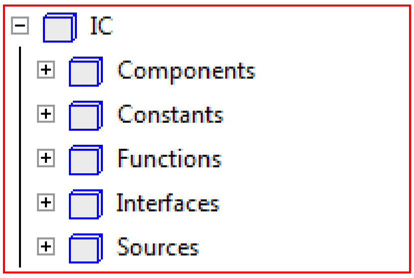

Figure 1: Library sub-packages

More detailed explanation of the models of each package is given in next chapter.

2.1 Components Sub-package

Components sub-package is the principal package of our library which contains all the components we have created except of voltage and current sources.
The components models included in this package are capacitor, ground, inductor, resistor, NMOS transistor and PMOS transistor.

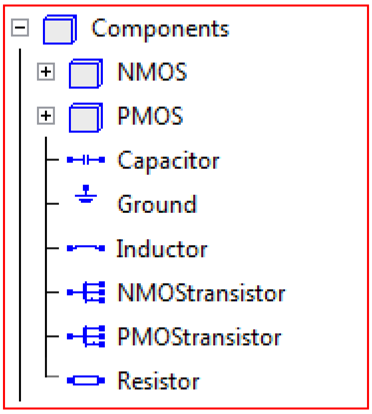

Figure 2: Components sub-package

Apart from these models, there are also two packages included which contains auxiliary models instantiated in the implementation of NMOS and PMOS transistors. The NMOS package contains the models used in NMOS transistor model implementation and PMOS package models used in PMOS transistor model implementation.

2.2 Constants Sub-package

Inside Constants sub-package we can find five different models. Each one contains constants and parameters necessary in components models except length and width which can be updated by the user.
In that way there is, for example, a model which contains constants for capacitor model like dielectric relative permittivity or separation between metal plates.

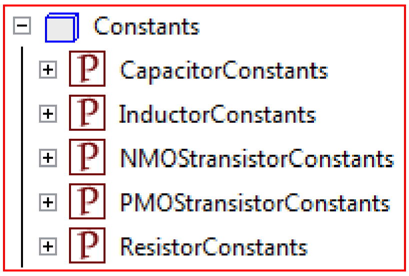

Figure 3: Constants sub-package

Constants included in these models should be changed before students use the library to start creating their circuits, making these constants transparent to them.

Students will only be able to edit length and width of each component.

The components which have a constants model associated are:

- Resistor
- Capacitor
- Inductor
- NMOS transistor
- PMOS transistor

2.3 Functions Sub-package

In Functions sub-package we have introduced some necessary functions to make some conversions used in components models such as conversion to decibels or from degrees to radians.

```
function Deg2Rad
  input nonSI.Angle_deg angle_deg;
  output SI.Angle angle_rad;
protected
  constant Real pi = 3.14159265358979;
algorithm
  angle_rad:=(pi * angle_deg) / 180;
```

Figure 4: Function Example

2.4 Interfaces Sub-package

In this sub-package we include common interfaces models. In each interface we define equations for variables common to all the components.

As Peter Fritzson defines in its book: "Modelica connectors are instances of connector classes, which define the variables that are part of the communication interface that is specified by a connector. Thus, connectors specify external interfaces for interaction.", [4].

Interfaces included in this package are extended by components. These interfaces are:

- Pin (Positive or Negative)
- TwoPin
- OnePort
- MOSFET

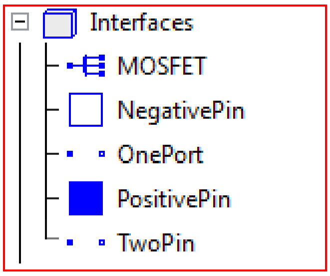

Figure 5: Interfaces sub-package

All these interfaces are partial models which means that they can not be used as models by themselves, they have to be extended by another model.

2.5 Sources Sub-package

After creating all the components of our library we need to create sources compatible with our components (i.e. using the same variables) to supply voltage or current to them. These sources are the models which make up this package.

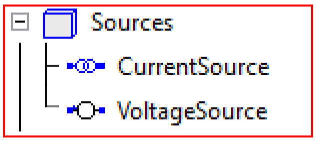

Figure 6: Sources sub-package

We have implemented two different sources, a voltage source and a current source. Both supply three different voltage or current variables which are: DC voltage or current, AC voltage or current and transient voltage or current. These voltages can be set, once the student has created the circuit, in the parameters option of the sources the student has added to the model.

2.6 Examples Sub-package

Examples sub-package contains some example models so intended users can simulate circuits which are already built and whose parameters have already been adjusted although user can change them if they need it.

3. Model Implementation

In chapter 3 we will explain in detail all the models contained in the sub-packages described in previous chapter. Here is only explained how models work, if you are interested in code go to Appendix A.

3.1 Functions Sub-package

Decibels Function

It is a very simple function which converts an input variable to its decibel value.

```
x_dB:= 20 * Math.log10(x);
```

Deg2Rad Function

This function is used to get the equivalent radians of a given degree.

```
angle_rad:=(pi * angle_deg) / 180;
```

Rad2Deg Function

The functionality of this function is the opposite to Deg2Rad function, a given radian is converted to degrees.

```
angle_deg:=(180 * angle_rad) / pi;
```

Rect2Polar Function

Given an array with x-coordinate and y-coordinate, this function returns amplitude and a phase value.

```
module:=sqrt(rect * rect);
angFirst:=if not module > 0 then 0 else Math.asin(abs(rect[2]) / module);
angle:=if rect[1] < 0 then pi - angFirst else angFirst;
angle:=if rect[2] < 0 then -angle else angle;
angle:=(180 * angle) / pi;
```

3.2 Constants Sub-package

Constants sub-package includes files with constants for most of the components of our library. These constants are just some of the constants you could find in the component models of other tools, but for our purpose, they are enough.

CapacitorConstants

Constants used in capacitor model are: dielectric relative permittivity ε_r (the ratio of the amount of electrical energy stored in the dielectric by an applied voltage, relative to that stored in a vacuum) and separation between the metal plates of the capacitor.

```
final constant Real Er = 3.4 "relative permittivity of capacitor dielectric";
final constant SI.Distance t = 5e-08 "Separation of capacitor metal plates";
```

ResistorConstants

Constant needed in our resistor model is the sheet resistance of the resistor.

```
final constant SI.Resistance Rs = 50 "Sheet resistance of the resistor
  material";
```

NMOStransistorConstants

In the implementation of the transistor models we have followed the model implemented in Modelica Library SPICELib [2], which implements SPICE Level 1 model. The constants needed in this model are shown below.

```
constant SI.Area AD = 0 "drain difussion area";
constant SI.Area AS = 0 "Source difussion area";
constant SI.Permittivity CGBO = 3e-10 "Gate-bulk overlap capacitance per
  meter [F/m]";
constant SI.Permittivity CGDO = 3e-10 "Gate-drain overlap capacitance per
  meter [F/m]";
constant SI.Permittivity CGSO = 3e-10 "Gate-source overlap capacitance per
  meter [F/m]";
constant Real CJ = 0.0001 "Capacitance at zero-bias voltage per square meter
  of area [F/m2]";
constant SI.Permittivity CJSW = 5e-10 "Capacitance at zero-bias voltage per
  meter of perimeter [F/m]";
constant Real FC = 0.5 "Substrate-junction forward-bias coefficient";
constant Real GAMMA = 0.4675 "Body-effect parameter [V0.5]";
constant SI.Current IS = 1e-14 "Reverse saturation current at 300K";
constant Real KP = 5e-05 "Transconductance parameter [A/V2]";
constant Real LAMBDA = 0.033 "Channel-length modulation [V-1]";
constant SI.Length LD = 0 "Lateral diffusion";
constant Real MJ = 0.75 "Bulk junction capacitnce grading coefficient";
constant Real MJSW = 0.33 "Perimeter capacitance grading coefficient";
constant SI.Length PD = 0.0004 "drain difussion perimeter";
```

```
constant SI.Length PS = 0.0004 "source difussion perimeter";
constant SI.Voltage PB = 0.95 "Junction potencial";
constant SI.Voltage PHI = 0.6 "Surface inversion potencial";
constant SI.Length TOX = 4.2e-08 "Gate oxide thickness";
constant SI.Voltage VTO = 0.43 "Zero-bias threshold voltage";
constant Real EPSR = 3.9 "Dielectric constant of the oxide";
```

PMOStransistorConstants

Constants of PMOS transistors are the same as NMOS transistor except threshold voltage (VTO) which should be negative.

```
constant SI.Voltage VTO = -0.43 "Zero-bias threshold voltage";
```

3.3 Interfaces Sub-package

Here are included interfaces models. We define in them, common variables to all the components

Pin

A Pin connector has across variables, e.g. voltages, and flow variables, e.g. currents, that means that the sum of all the currents which arrive to the pin must be equal to the currents wich go out from it.
We have four kind of voltage or current variables: DC, Transient and real and imaginary part of AC voltage or current.

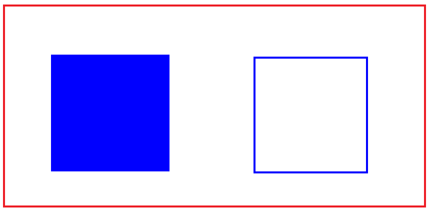

Figure 7: Pins icons

We have to kind of pins, a positive pin type and negative pin type. The only difference between them is the color (postive: blue, negative: white)

These are a partial model so other models have to instantiate them.

```
SI.Voltage vDC "DC potential at the pin";
SI.Voltage vTran "Transient/Small-signal potential at the pin";
SI.Voltage vAC_Re "Small-signal potential at the pin. Real part";
SI.Voltage vAC_Im "Small-signal potential at the pin. Imaginary part";
flow SI.Current iDC "DC current flowing into the pin";
flow SI.Current iTran "Transient current flowing into the pin";
flow SI.Current iAC_Re "Small-signal current flowing into the pin. Real
  part";
flow SI.Current iAC_Im "Small-signal current flowing into the pin. Imaginary
  part";
```

Two Pin

This is also a partial model (it means that it can not be used as a model by itself, it must be instantiate by another model) which intantiate two Pin models, so a TwoPin interface consist of two pins. If we want to define, for example, voltages betwwen these two pins we have to do it in the model which instantiate TwoPin interface.

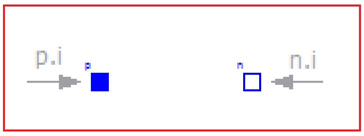

Figure 8: TwoPin icon

```
PositivePin p "(+) node";
NegativePin n "(-) node";
```

OnePort

This model extends, i.e. inherits all the variables, parameters and equations of TwoPin, so it is also made up of two pins, and we define DC, AC and transient variables as the difference between each pin variables.

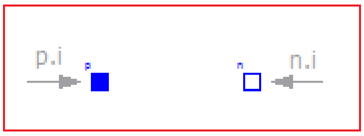

Figure 9: OnePort icon

```
extends TwoPin;
  SI.Voltage vDC "DC voltage between pines";
  SI.Voltage vTran "Transient voltage between pines";
  SI.Voltage vAC_Re "Real part of AC small-signal voltage between pines";
  SI.Voltage vAC_Im "Imaginary part of AC small-signal voltage between pines";
  SI.Current iDC "DC current";
  SI.Current iTran "Transient/Small-signal current";
  SI.Current iAC_Re "Small-signal current. Real part";
  SI.Current iAC_Im "Small-signal current. Imaginary part";

equation
  {iDC,iTran,iAC_Re,iAC_Im} = {p.iDC,p.iTran,p.iAC_Re,p.iAC_Im};
  {iDC,iTran,iAC_Re,iAC_Im} = -{n.iDC,n.iTran,n.iAC_Re,n.iAC_Im};
  {vDC,vTran,vAC_Re,vAC_Im} = {p.vDC,p.vTran,p.vAC_Re,p.vAC_Im} -
  {n.vDC,n.vTran,n.vAC_Re,n.vAC_Im};
```

MOSFET

This will be the interface extended by NMOS and PMOS transistors. It makes four Pin instances , one for each pin of a MOS transistor (gate, source, drain and bulk).

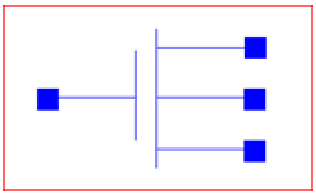

Figure 10: MOSFET icon

```
PositivePin d "Drain";
PositivePin s "Source";
PositivePin g "Gate";
PositivePin b "Bulk";
```

3.4 Sources Sub-package

CurrentSource

This is the model which will supply current to our circuits. For that purpose we define several parameters which can be edited by users to control current variables (DC, AC, transient). Voltage variables will be defined by the circuit.

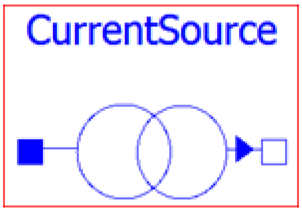

Figure 11: CurrentSource icon

For AC current we have to edit "Amplitude" and "Phase".

```
parameter SI.Current Amplitude(start = 1) "Amplitude AC signal";
parameter nonSI.Angle_deg phase = 0 "Phase of AC signal";
```

For DC current we have "DCCurrent" parameter.

```
parameter SI.Current DCcurrent = 1e-05 "DC offset";
```

Transient current is a bit more complicated. The idea is to create a pulse. For that we have to define a delay, a rise time, a step width and an initial current and final current.

```
parameter Real delay = 0.1 "Time (in microsecs) when transient step rises";
parameter Real riseTime = 50 "Time (in nanosecs) which takes signal going
  from V0 to V1";
parameter Real fallTime = 50 "Time (in nanosecs) which takes signal going
  from V1 to V0";
parameter Real stepWidth = 200 "Time (in nanosecs) since signal reachs V1
  until it start to fall towards V0";
parameter SI.Current I0 = 0 "Low level current of the step";
parameter SI.Current I1 = 1 "High level current of the step";
```

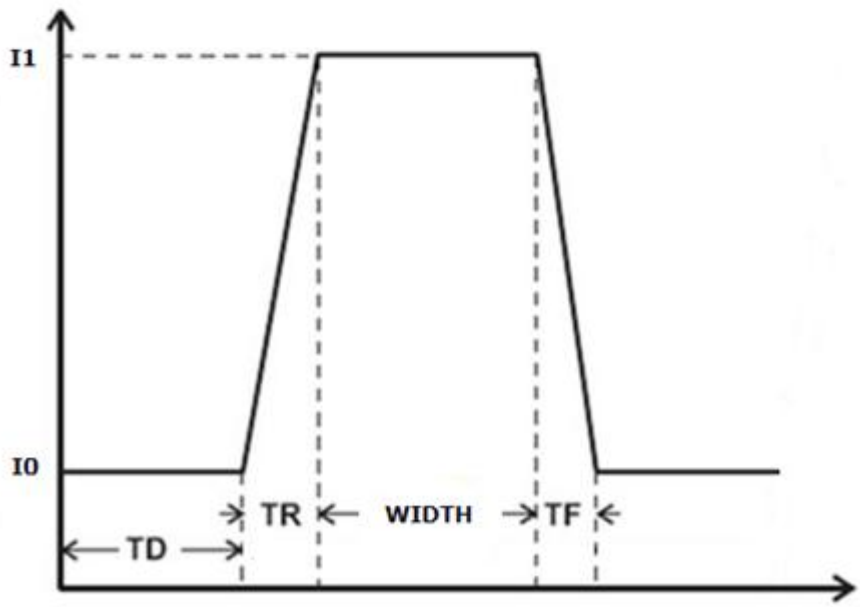

Figure 12: Pulse parameters diagram

Apart from these parameters we define some useful variables such as iAC_mag_dB which is the magnitude of AC current measured in decibels.

```
SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage";
SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
  across the component";
SI.Conversions.NonSIunits.Angle_deg vAC_phase(start = 0) "Phase (deg) of AC
  small-signal voltage";
SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal current";
SI.Voltage pinP_iAC_mag(start = 0) "Magnitude of AC small-signal current at
  positive Pin";
nonSI.Angle_deg pinP_iAC_phase(start = 0) "Phase (deg) of AC small-signal
  current at positive Pin";
SI.Voltage pinP_iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
  current at positive Pin";
SI.Voltage pinN_iAC_mag(start = 0) "Magnitude of AC small-signal voltage at
  negative Pin";
nonSI.Angle_deg pinN_iAC_phase(start = 0) "Phase (deg) of AC small-signal
  current at negative Pin";
SI.Voltage pinN_iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
  current at negative Pin";
```

And finally, we set the equations which relate parameters with variables created in this model and ones heritated from OnePort interface.

```
iDC = DCcurrent;
iAC_mag = Amplitude;
iAC_mag_dB = Decibels(vAC_mag);
{iAC_Re,iAC_Im} = {Amplitude * cos(Deg2Rad(phase)),Amplitude *
  sin(Deg2Rad(phase))};
(vAC_mag,vAC_phase) = Rect2Polar({vAC_Re,vAC_Im});
vAC_mag_dB = Decibels(vAC_mag);
(pinP_iAC_mag,pinP_iAC_phase) = Rect2Polar({p.iAC_Re,p.iAC_Im});
```

```
pinP_iAC_mag_dB = Decibels(pinP_iAC_mag);
(pinN_iAC_mag,pinN_iAC_phase) = Rect2Polar({n.iAC_Re,n.iAC_Im});
pinN_iAC_mag_dB = Decibels(pinN_iAC_mag);
assert(riseTime + fallTime < stepWidth, "Rise time + Fall time is bigger
  than step width in current source parameters");
iTran = if time < delay * 1e-06 then I0 else if time > delay * 1e-06 and
  time < delay * 1e-06 + riseTime * 1e-09 then (I1 - I0) / riseTime * 1e-09
  *(time - delay * 1e-06) else if time > delay * 1e-06 + riseTime * 1e-09
  and time < delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 then I1
  else if time > delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 and
  time < delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 + fallTime *
  1e-09 then I1 - (I1 - I0) / fallTime * 1e-09 * (time - delay * 1e-06 +
  riseTime * 1e-09 + stepWidth * 1e-09) else I0;
```

<u>VoltageSource</u>

In this case we can edit the voltage variables through parameters, but the idea is the same as in current source, setting a DC voltage, an amplitude and phase of AC voltage and delay, rise time, fall time and step width for transient voltage.

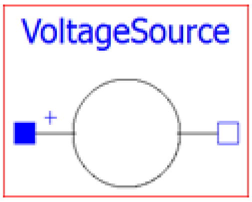

Figure 13: VoltageSource icon

```
parameter SI.Voltage Amplitude(start = 1) "Amplitude AC signal";
parameter nonSI.Angle_deg phase = 0 "Phase of AC signal";
parameter SI.Voltage DCVoltage = 1.2 "DC offset";
parameter Real delay = 0.1 "Time (in microsecs) when transient step rises";
parameter Real riseTime = 50 "Time (in nanosecs) which takes signal going
  from V0 to V1";
parameter Real fallTime = 50 "Time (in nanosecs) which takes signal going
  from V1 to V0";
parameter Real stepWidth = 200 "Time (in nanosecs) since signal reachs V1
  until it start to fall towards V0";
parameter SI.Voltage V0 = 0 "Low level voltage of the step";
parameter SI.Voltage V1 = 1 "High level voltage of the step";
```

We define also useful variables which can give us more precise information.

```
SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage";
SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
  across the component";
SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal current";
SI.Conversions.NonSIunits.Angle_deg iAC_phase(start = 0) "Phase (deg) of AC
  small-signal current";
SI.Voltage pinP_vAC_mag(start = 0) "Magnitude of AC small-signal voltage at
  positive Pin";
nonSI.Angle_deg pinP_vAC_phase(start = 0) "Phase (deg) of AC small-signal
  voltage at positive Pin";
SI.Voltage pinP_vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
  voltage at positive Pin";
SI.Voltage pinN_vAC_mag(start = 0) "Magnitude of AC small-signal voltage at
  negative Pin";
nonSI.Angle_deg pinN_vAC_phase(start = 0) "Phase (deg) of AC small-signal
  voltage at negative Pin";
SI.Voltage pinN_vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
  voltage at negative Pin";
```

And set the governing equations related to voltage source.

```
vDC = DCVoltage;
vAC_mag = Amplitude;
vAC_mag_dB = Decibels(vAC_mag);
{vAC_Re,vAC_Im} = {Amplitude * cos(Deg2Rad(phase)),Amplitude *
  sin(Deg2Rad(phase))};
(iAC_mag,iAC_phase) = Rect2Polar({iAC_Re,iAC_Im});
iAC_mag_dB = Decibels(iAC_mag);
(pinP_vAC_mag,pinP_vAC_phase) = Rect2Polar({p.vAC_Re,p.vAC_Im});
pinP_vAC_mag_dB = Decibels(pinP_vAC_mag);
(pinN_vAC_mag,pinN_vAC_phase) = Rect2Polar({n.vAC_Re,n.vAC_Im});
pinN_vAC_mag_dB = Decibels(pinN_vAC_mag);
assert(riseTime + fallTime < stepWidth, "Rise time + fall time is bigger
  than step width in voltage source parameters");
vTran = if time < delay * 1e-06 then V0 else if time < delay * 1e-06 +
  riseTime * 1e-09 then V0 + (V1 - V0) / (riseTime * 1e-09) * (time - delay
  *1e-06) else if time < delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-
  09 then V1 else if time < delay * 1e-06 + riseTime * 1e-09 + stepWidth *
  1e-09 + fallTime * 1e-09 then V1 - (V1 - V0) / (fallTime * 1e-09) * (time
  - delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09) else V0;
```

3.5 Components Sub-package

Now we will see more in detail each of our components.

Ground

This component is needed in any circuit we want to built up because it sets ground. It is a really simple component which instantiate a Pin interface and the equations say that DC, AC and transient voltages are set to zero.

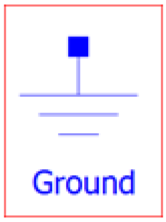

Figure 14: Ground icon

```
{p.vDC,p.vTran,p.vAC_Re,p.vAC_Im} = zeros(4);
```

Resistor

We have used as reference the wire resistor model from [7].
We can think about the resistor as a brick with length L, width W and thickness t.
So total resistance can be defined as:

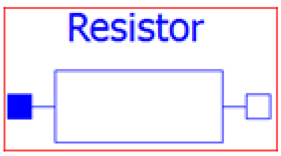

Figure 15: Resistor icon

$$R = \frac{\rho \cdot L}{t \cdot W} \qquad (1)$$

Where ρ represents resistivity of the material, L the length, W the width and t the thickness of the resistor. In equation [1] we can define sheet resistance as:

$$R_s = \frac{\rho}{t} \quad (2)$$

So, finally we have:

$$R = \frac{R_S \cdot L}{W} \quad (3)$$

That is why the only constant necessary for resistor is sheet resistance. Then users just need to edit width and length to give a value of resistance.

The equations of resistor are similar to all variable types (DC, AC, transient) and it is:

$$V = R \cdot I \quad (4)$$

That is the famous Ohm's law.

<u>Capacitor</u>

In this component we have followed a parallel plates model which is the kind of capacitors used in integrated circuits, [8]. This capacitors are made up by two parallel metal plates separated by a dielectric. The parameters we have are: the width W and the length L of metal plates, the separation t between plates and relative permittivity ε_r of the dielectric.
So, total capacitance would be:

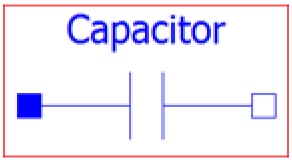

Figure 16: Capacitor icon

$$C = 0{,}0885 \cdot \frac{W \cdot L \cdot \varepsilon}{t} \quad (5)$$

where W and L should be expressed in micrometers and t in Angstroms to obtain capacitance measured in picoFarads.

In the case of capacitors, equations change depending on the type of variables considered.

- In DC the equation is:

$$iDC = 0 \quad (6)$$

- In AC case:

$$vAC = \frac{1}{j2\pi fC} \cdot iAC \qquad (7)$$

- For transient variables:

$$iTran = C \cdot \frac{dvTran}{dt} \qquad (8)$$

<u>Inductor</u>

Although inductors are not so normally used in simple integrated circuits configurations as capacitors or resistors just in case it is needed. It is why the only parameter of this model is inductance, to make it really simple.

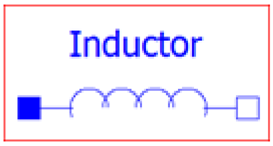

Figure 17: Inductor icon

The equations of inductor are similar to capacitor equations but only switching voltage and current variables.

$$iDC = 0 \qquad (9)$$

$$vAC = j2\pi fL \cdot iAC \qquad (10)$$

$$vTran = L \cdot \frac{diTran}{dt} \qquad (11)$$

Inductance (L) value can be edited in properties window. (See chapter 4)

<u>NMOS and PMOS transistors</u>

For the implementation of both NMOS and PMOS transistor we have followed SPICELib Modelica Library [2] model which is SPICE LEVEL 1 model, [9]. It is a standard model for circuits simulation developed at the Electronics Research Laboratory of the University of California, Berkeley.

For the implementation we consider two equivalent models, one for large signal, from which we obtain DC and transient variables, and one for small signals from which we obtain AC variables, obtaining in that way the three typical analysis we use to do to a circuit.

Large signal equivalent model looks as follows:

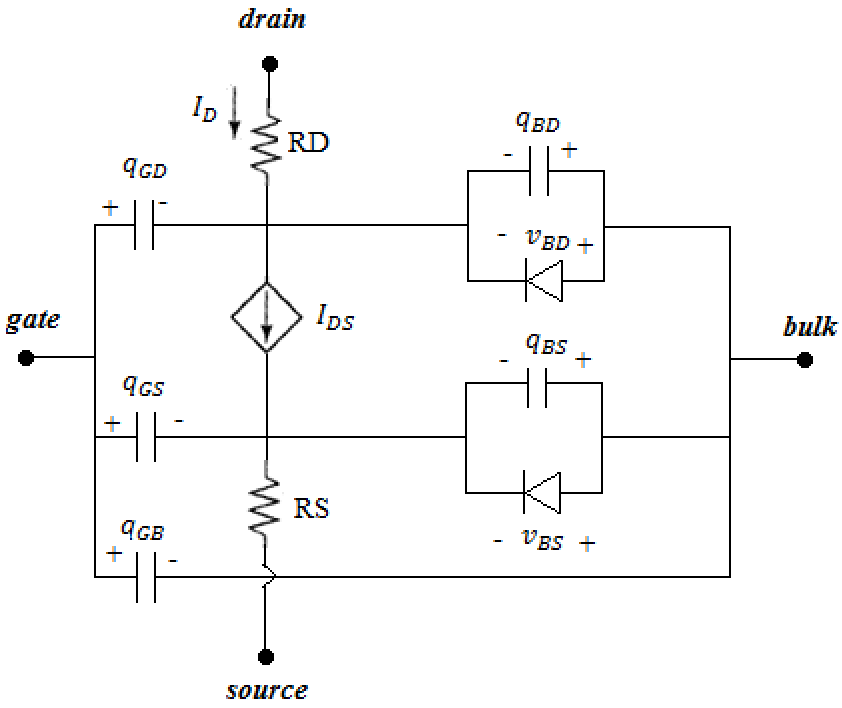

Figure 18: NMOS transistor large signal equivalent model

Q_{gd}, Q_{gs} and Q_{gb} represent the overlap capacitance of the gate with drain, source and bulk connection area.
R_d and R_s represent impedance of drain and source.
Diodes and capacitors represent PN unions between substrate and drain and source contact areas.
And I_{ds} represents the current through the transistor. This current behavior is governated by following equations which depends on voltage difference between different pins of the transistor.

$$I_{DS} = 0 \qquad (V_{GS} \le V_{TH})$$

$$I_{DS} = \frac{KP}{2}(W/L_{eff})V_{DS}[2(V_{GS} - V_{TH}) - V_{DS}](1 + LAMBDA \cdot V_{DS}) \qquad (0 \le V_{DS} \le V_{GS} - V_{TH})$$

$$I_{DS} = \frac{KP}{2}(W/L_{eff})(V_{GS} - V_{TH})^2(1 + LAMBDA \cdot V_{DS}) \qquad (0 \le V_{GS} - V_{TH} \le V_{DS})$$

All the components described before are included in NMOS or PMOS packets and are instantiated from NMOS and PMOS transistor models. All the

parameters of this components appear in NMOS or PMOS constants file in Constants sub-package.
And small signal equivalent model:

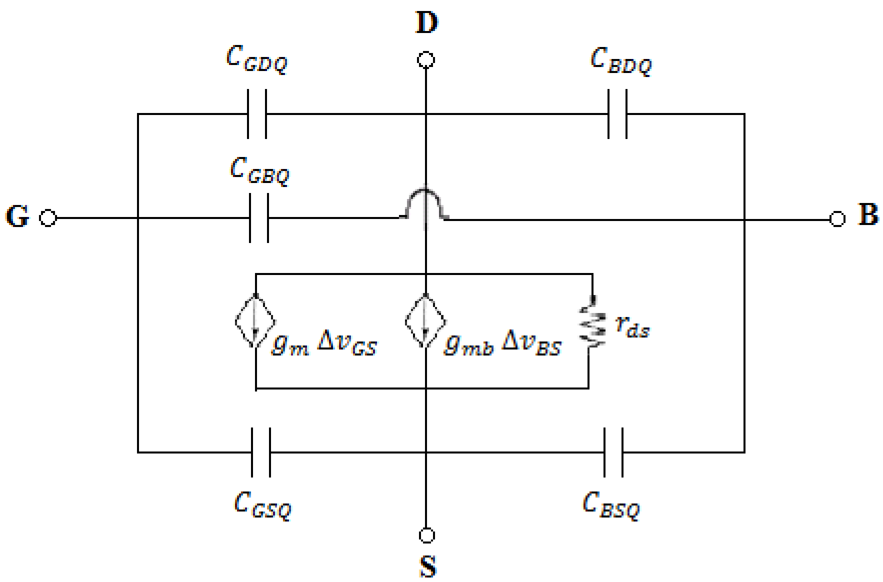

Figure 19: NMOS transistor small-signal equivalent model

In the case of small signal equivalent model, the current source is modeled as two current sources with a resistor in parallel. The current of these sources depends on, as in the circuit before, of voltages between transistor pins but also on DC current calculated in the large signal model.

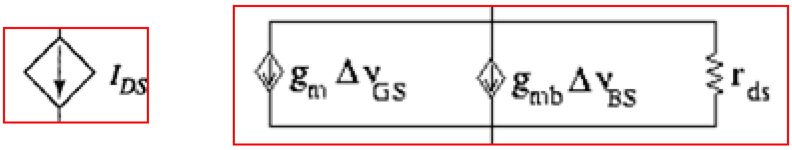

Figure 20: Transistor current models in a) large and b) small signal models

So, when we simulate this model and DC, AC and transient variables are calculated we are obtaining DC, AC and transient analysis, without having to do anything else, as it is needed in SPICELib library, [10].

The only difference between PMOS and NMOS transistor is that in NMOS transistor we consider three principal voltages:

- V_{gs} (voltage in the gate minus voltage in the source)
- V_{ds} (voltage in the drain minus voltage in the source)
- V_{bs} (voltage in the bulk minus voltage in the source)

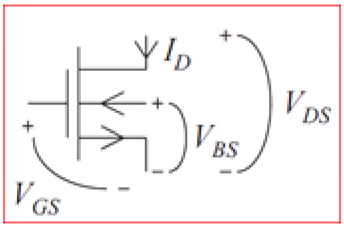

Figure 21: NMOS variables [11]

While in PMOS transistor we consider:

- V_{sg} (voltage in the source minus voltage in the gate)
- V_{sd} (voltage in the source minus voltage in the drain)
- V_{sb} (voltage in the source minus voltage in the bulk)

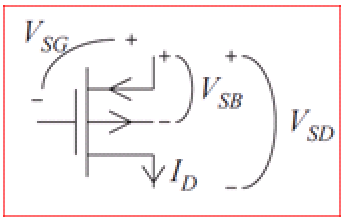

Figure 22: PMOS variables [11]

And therefore, voltage threshold parameter is considered negative in PMOS transistor.

3.6 Examples Sub-package

In this package we can find some circuit models already prepared for the intended users. These models are some simple and usual circuits such as inverters or amplifiers stages, so students can simulate them and observe the results.

In Chapter 5 (Test Cases) we explain in detail how to simulate these models and we check, looking at different variables, that these models behave as the components they represent.

The models included in this package are:

- CMOSInverter:

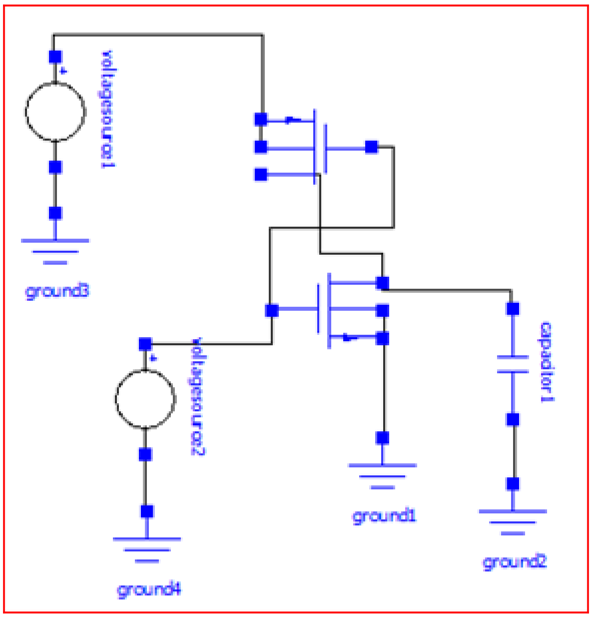

Figure 23: CMOS inverter configuration

It represents an inverter implemented with CMOS technology.
When we have a high level in voltagesource2, PMOS transistor cut-off, acting like high impedance, and NMOs transistor in "ON" acting as low impedance, so we have ground voltage in the output.
When we have a low level in voltagesource2, it is the opposite, PMOS transistor is "ON" and NMOS transistor is "OFF", getting supply voltage in the output.

NMOS transistor is "ON" when its gate-source voltage is higher than V_t(high level of voltagesource2), and PMOS transistor is ON when its source gate is higher than V_t (low level in voltagesource2).

- Common drain stage

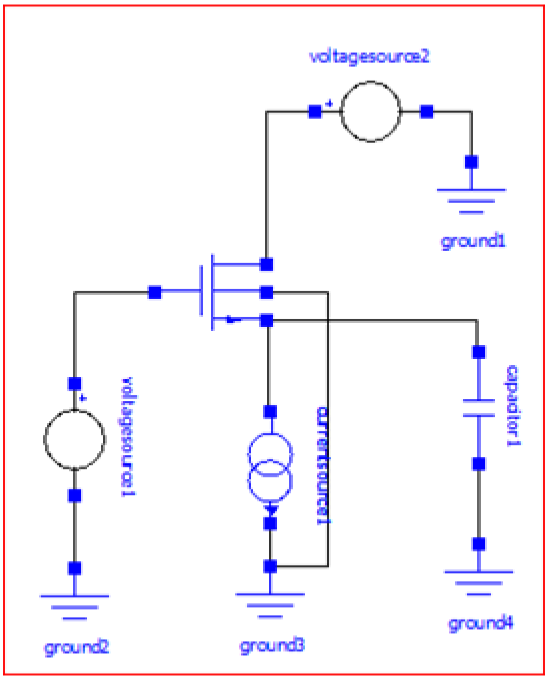

Figure 24: Common drain stage configuration

Our input voltage is connected to gate and output is collected in source being drain common to both, that is why its name. Source voltage should follow input voltage so gain of this circuit should be near to one.

If we analize the circuit, we can see that if the input (voltagesource1) increases, current through the transistor increases too, charging then the capacitor in the output which makes output voltage to increase. If input voltage decreases current decrease too, the capacitor discharges through currentsource1 and outputvoltage falls down.

So, we can see that output voltage follows input voltage.

- Common source stage

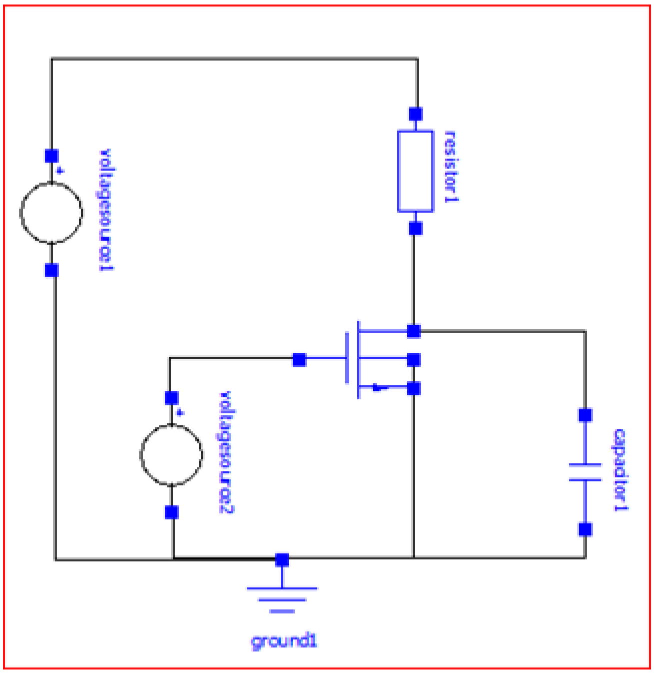

Figure 25: Common Source Stage configuration

In this case, we have input in gate pin and output in drain pin, being source common to both. We can obtain a good gain with this configuration (more than 10 dB)

- Current mirror

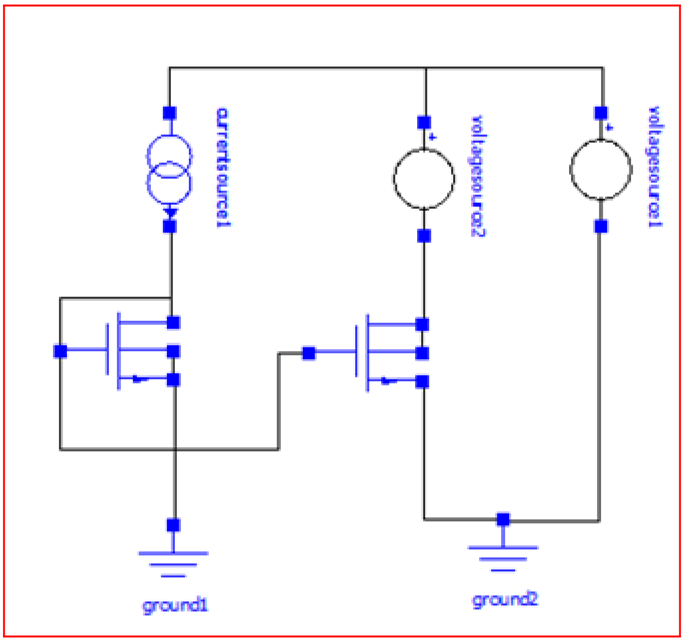

Figure 26: Current Mirror configuration

The functionality of this circuit is to copy current through a branch to the other one. It is done by connecting both transistors gates.
With currentsource1 we force a current through first NMOS transistor, forcing then its gate-source voltage. Since its gate is connected to the gate of the other transistor and both sources are connected to ground, they have the same value of gate-source voltage, producing then, the same current through the second NMOS transistor.

- Transmission gate

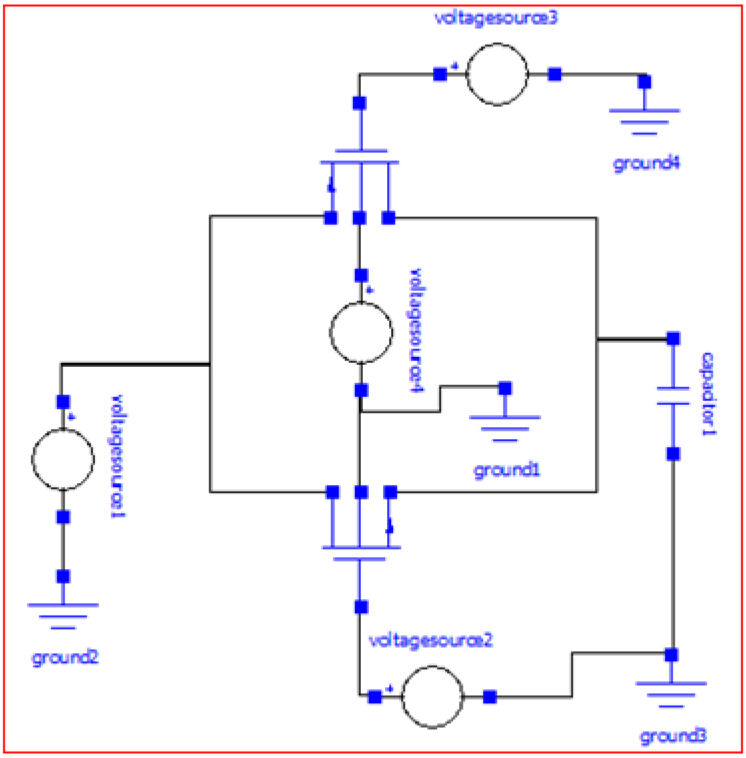

Figure 27: Transmission Gate configuration

This configuration is used as a switch, permiting to copy input voltage to the output always that NMOS transistor is “ON” and PMOS transistor “OFF”. When the voltage on node A is a logical “1”, the complementary logical “0” is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a logical “0”, the complementary logical “1” is applied to node A, turning both transistors off and forcing a high-impedance condition in both the IN and OUT nodes.

- Two Stage

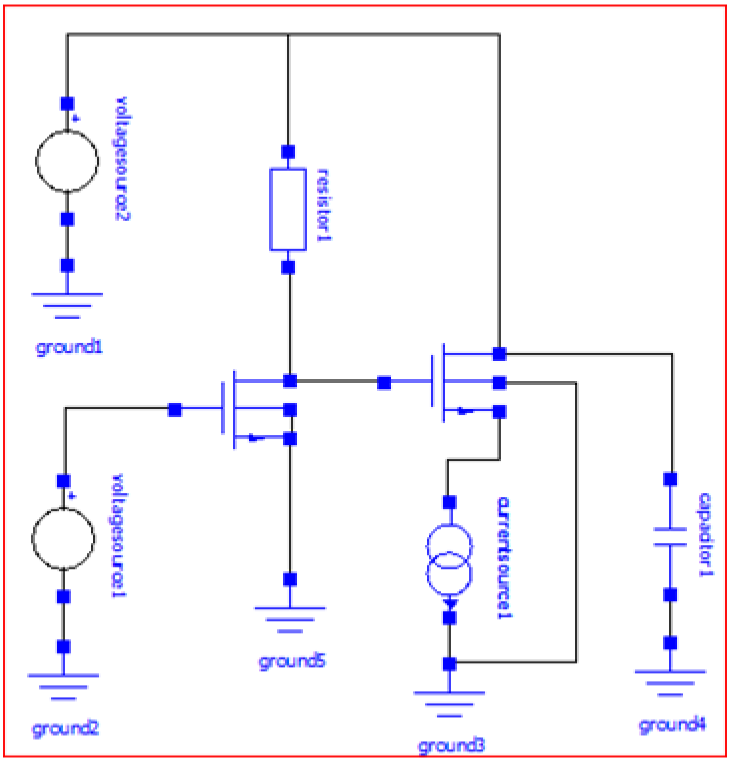

Figure 28: Two Stages configuration

Here we just connect in cascade two of the previous amplifiers, a common source and a common drain.

4. Getting Started

In this section we will explain how to use our library and how to create and simulate a simple circuit. For that purpose we will explain how we have built up CommonSourceStage example. This model will be created using OMEdit, the graphical editor of OpenModelica.

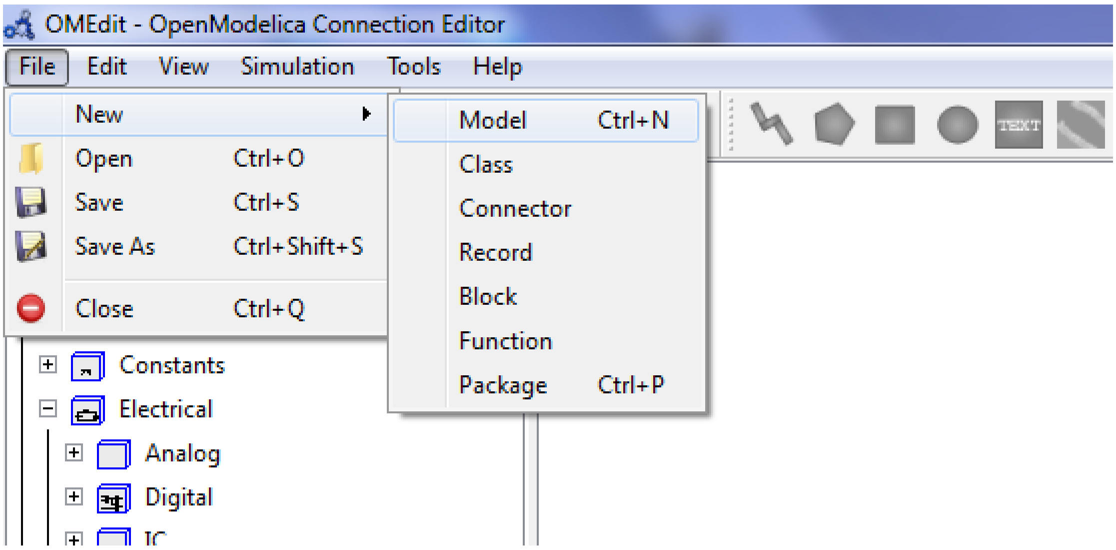

Figure 29: Creating a new model

4.1 Drag and Drop Components

First of all, we create a new file (File -> New -> Model) and give it a name.

Once we have created our model, we have to drag and drop all the components needed in our circuit. For doing that we just open our library, expand the components package and do click over the components and drag them to the white square which represents our model as it is shown in figure 30. In our case we need: a resistor, a capacitor, a NMOS transistor, a ground and two voltage sources.

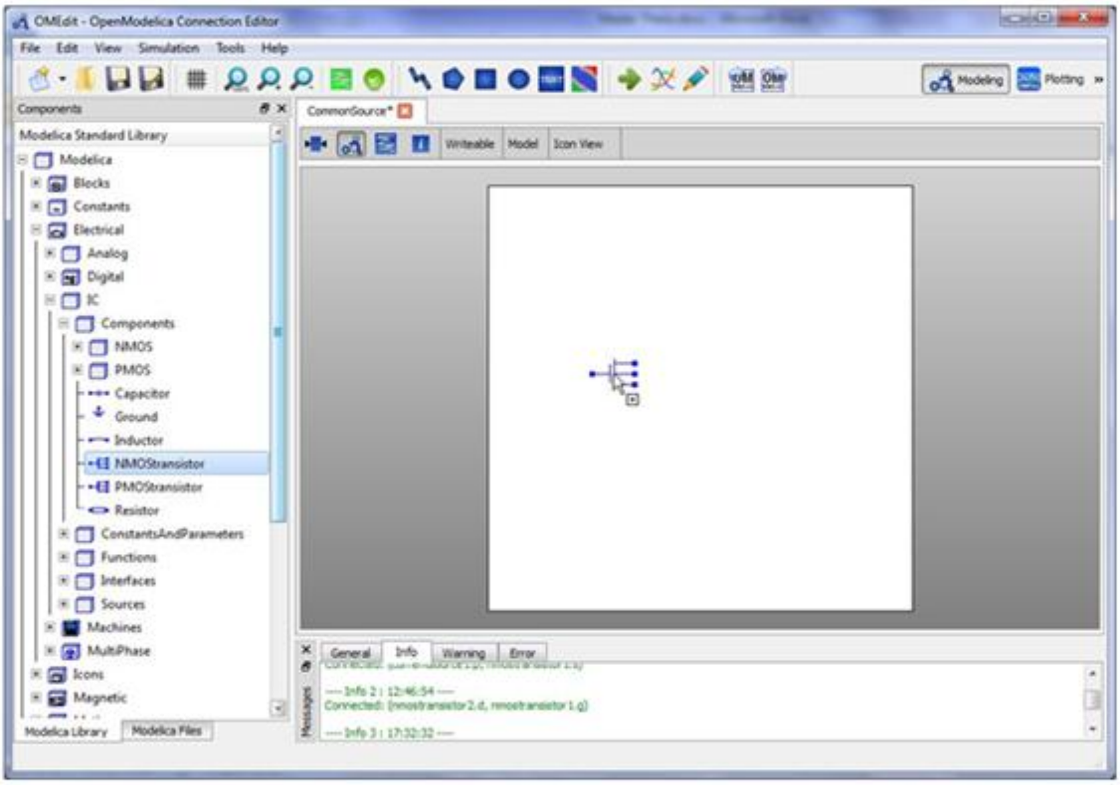

Figure 30: Dragging a component

You can also rotate components to place them properly just by right clicking over the component you want to rotate and selecting "Rotate Clockwise" or "Rotate Anticlockwise", see figure 31.

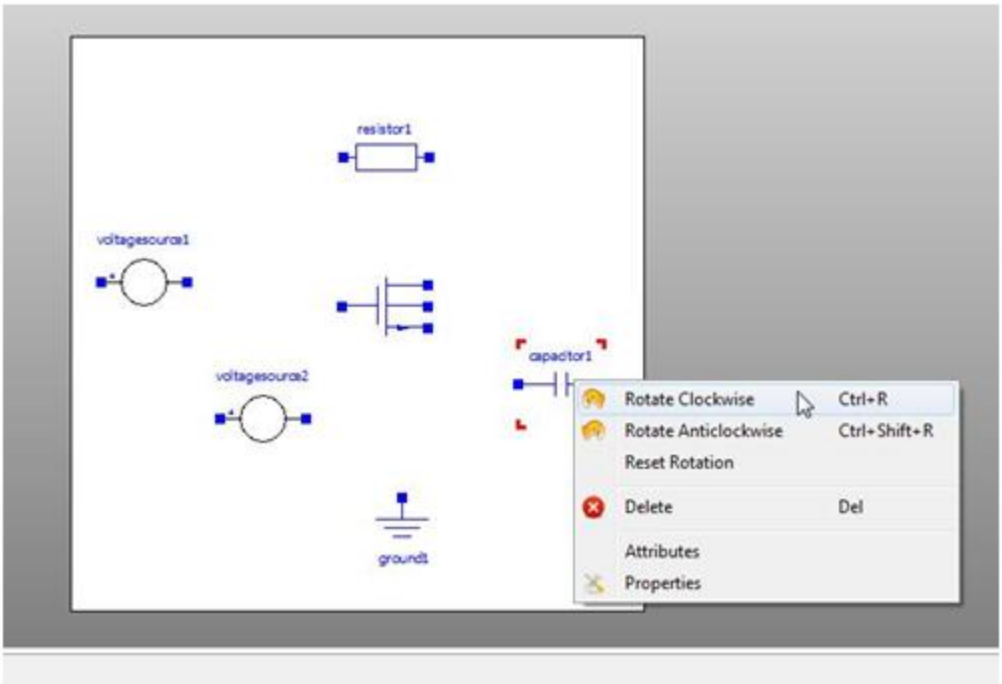

Figure 31: Rotate a component

4.2 Make Connections

After placing all our components we will make the connections between them. All you have to do is to click in one of the pins you want to connect and then click in the other pin. You will see a line which starts in the first pin and follows your mouse as shown in figure 32. If you want to make the line to turn just make another click in any place.

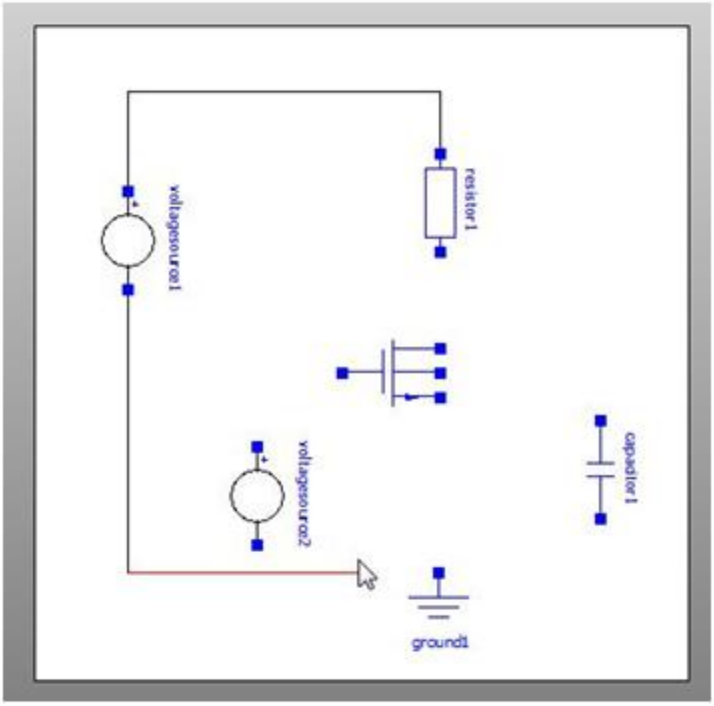

Figure 32: Connecting components

And finally you should have something like this:

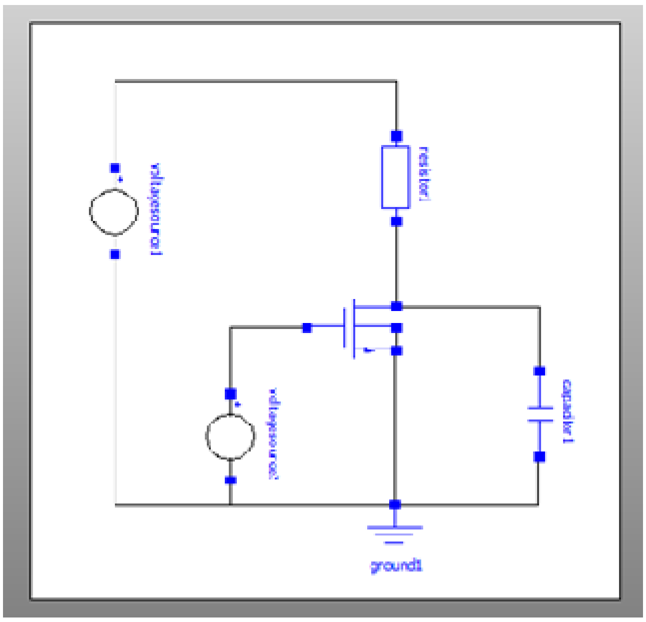

Figure 33: Complete model

4.3 Setting The Parameters

Now we have our circuit prepared and ready for editing its parameters.
What we have to do is right click in each of our components and open the "Properties" window. A new window will appear and if we select "Parameters" tab we can modify the parameters of that component.

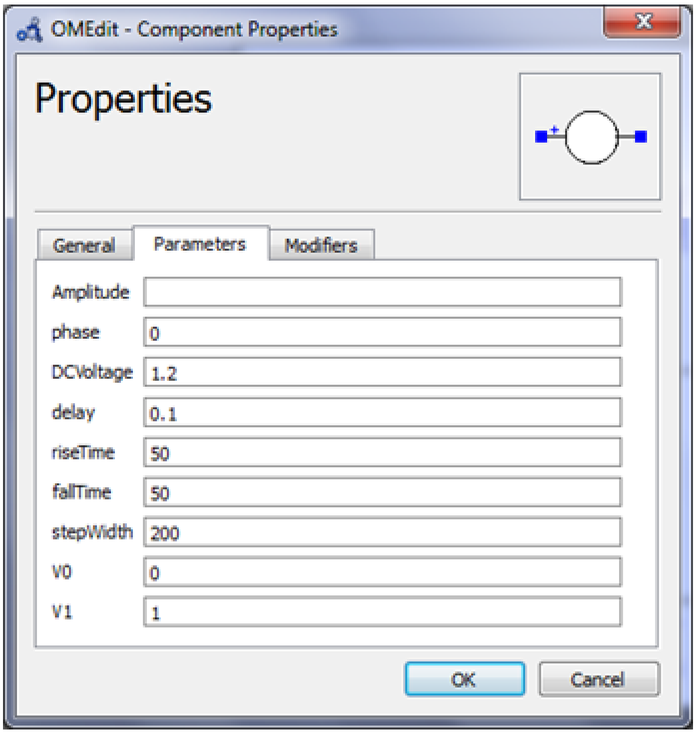

Figure 34: Properties window

4.4 Simulate

Now we have all the parameters updated. Before being able to simulate there is one more thing left. We have to select "Modelica Text View" in OMEdit so we can see the code of our circuit.

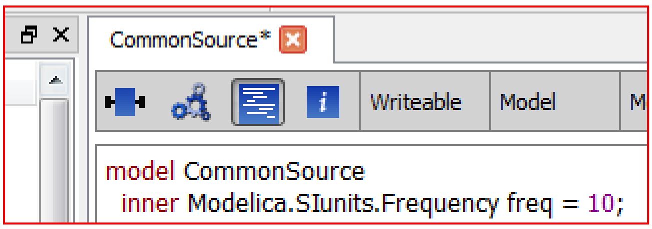

Figure 35: Inner sentence

And after that we have to insert the sentence “**inner Modelica.SIunits.Frequency freq = XX;**” just after the lines which says “**model NameOfYourModel**” (in this case “model CommonSource”). The XX value will be the value we will assign to the frequency of the circuit (We have assigned a value of 10 Hertz). See figure 35.

Now we are prepared to simulate our circuit. There is a simulation button at the top of the window.

Figure 36: "Simulate" icon

If we press it, a window will appear with all the simulation options, where we can set, for example, the start and stop time of the simulation. We press “Simulate!” and our model will be simulated. The time it takes depends on the complexity of our model, being possible to take a bit more of a minute for very complex models.

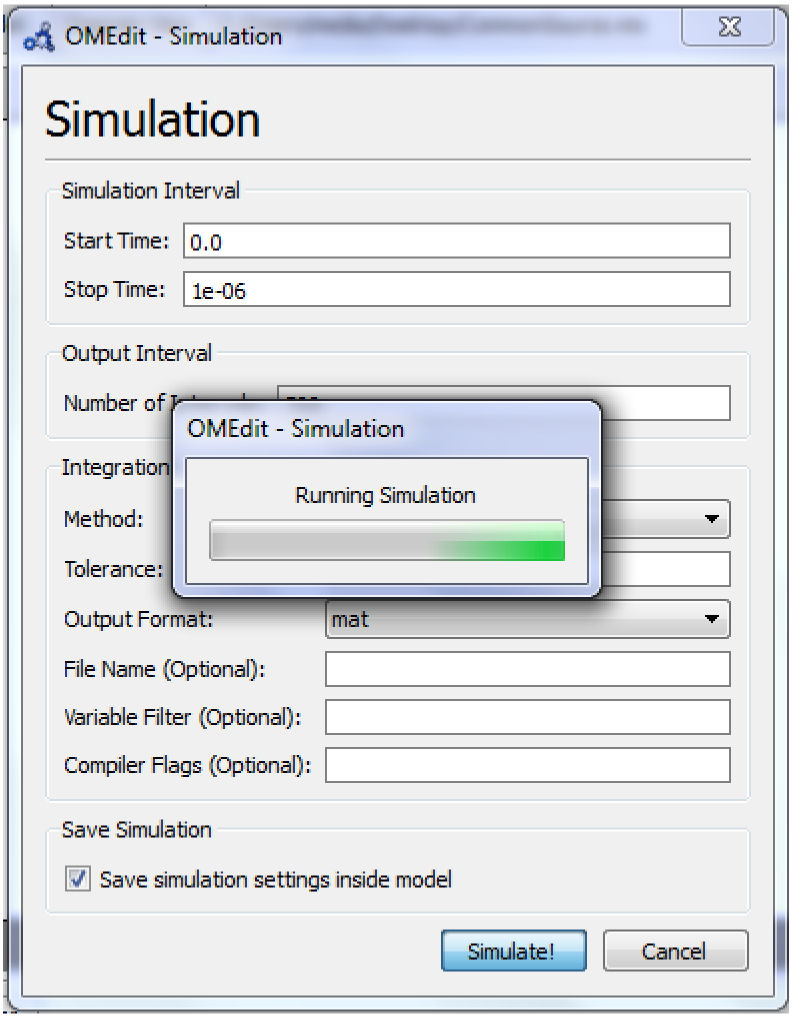

Figure 37: Simulation Dialog

When the simulation is done, we will be able to see the variables in the right part of the window and select them to be plotted. We can select as many variables as we want.

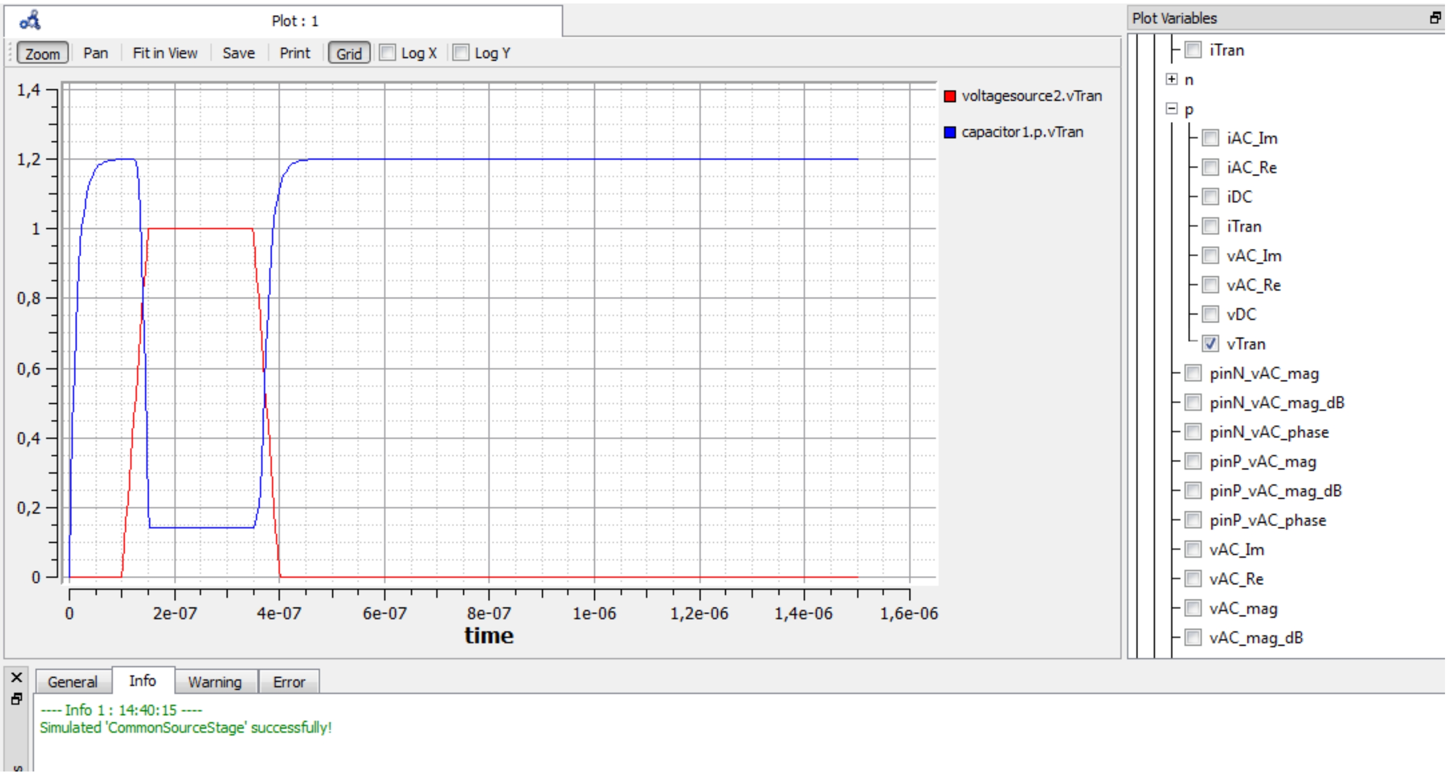

Figure 38: Plotting view

5. Test Cases

In this section we will explain each of the models which appear in Examples packages and observe their behavior as a test for our library. To be able to see them you have to open examples package since it is not included in Modelica Library so that you can change parameters already set.

CMOS Inverter

The composition of a PMOS transistor creates low resistance between its source and drain contacts when a low gate voltage is applied and high resistance when a high gate voltage is applied. On the other hand, the composition of an NMOS transistor creates high resistance between source and drain when a low gate voltage is applied and low resistance when a high gate voltage is applied.

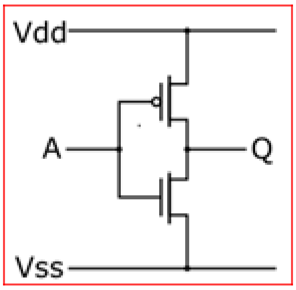

Figure 39: CMOS Inverter model

CMOS accomplishes current reduction by complementing every nMOSFET with a pMOSFET and connecting both gates and both drains together, [12].

A high voltage on the gates will cause the nMOSFET to conduct which provokes a voltage in the output Q equal to ground and the pMOSFET not to conduct while a low voltage on the gates causes the reverse making the voltage of the output follow supply voltage.

So, the interesting variables are transient variables. To check if our model is working properly we compare transient voltage of the voltage source connected to the gates of the transistors and voltage in the output which in this case is same as the transient voltage in the load capacitor. These voltages should be opposite, i.e., voltage in the capacitor should be a logical 1 when voltage in the source is 0 and vice versa.

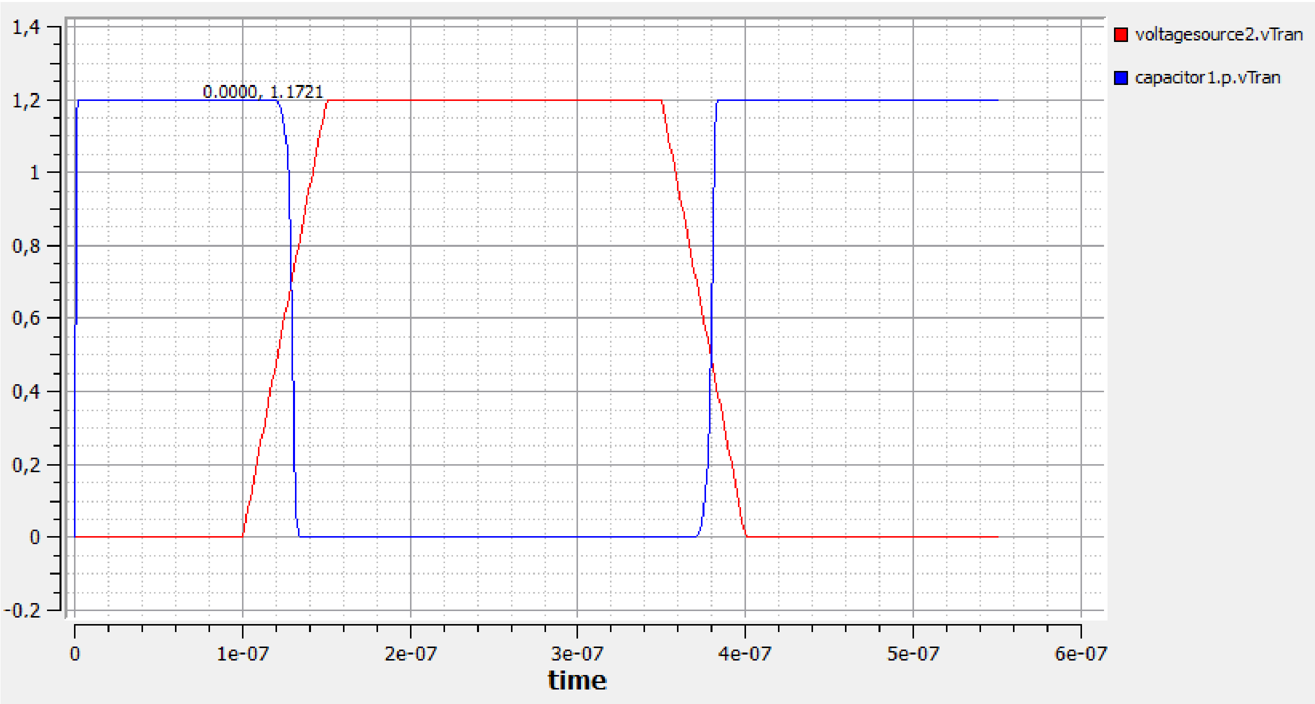

Figure 40: CMOSInverter Plot

<u>**Common Drain Stage**</u>

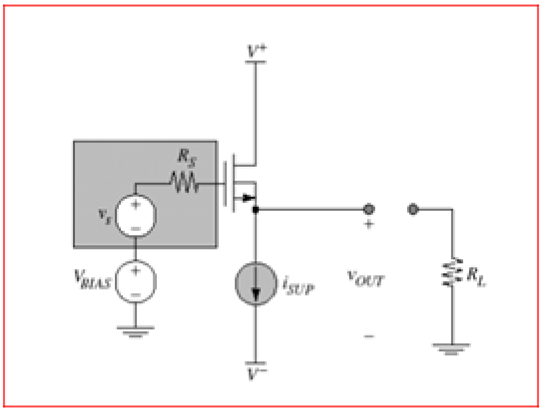

Figure 41: Common Drain Stage model

A common-drain amplifier, also known as a **source follower**, is one of three basic amplifier topologies (common drain, common source, common gate), typically used as a voltage buffer. In this circuit the gate terminal of the transistor serves as the input, the source is the output, and the drain is common to both (input and output), hence its name.

In addition, this circuit is used to transform impedances. For example, the Thévenin resistance of a combination of a voltage follower driven by a voltage source with high Thévenin resistance is reduced to only the output resistance of the voltage follower, a small resistance. That resistance reduction makes the combination a more ideal voltage source. Conversely, a voltage follower inserted between a driving stage and a high load (i.e. a low resistance) presents an infinite resistance (low load) to the driving stage, an advantage in coupling a voltage signal to a large load, [13].

As a source follower, source voltage follows gate voltage giving a gain near to 1 (0 dB).

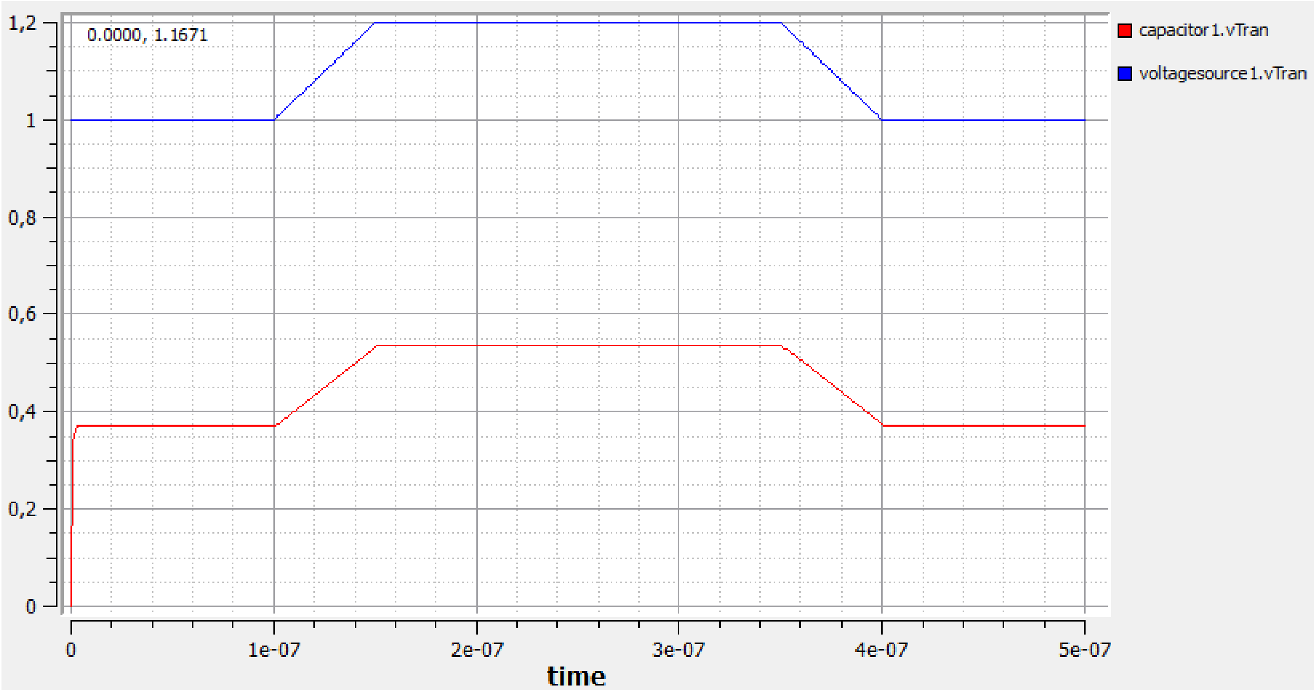

Figure 42: Common drain stage plot

We observe transient voltage in gate and source to check that the latter is following the former.

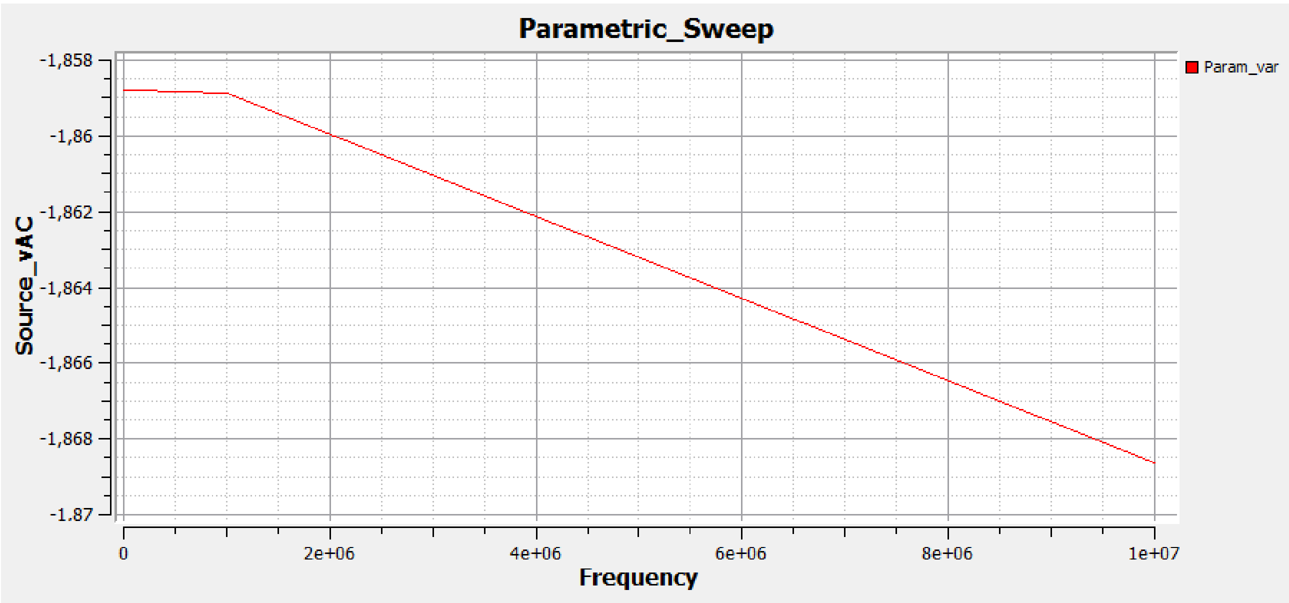

Figure 43: Common Drain Stage Parametric Sweep

If we use python program described in chapter 6 to make a parametric sweep of frequency we can obtain a frequency response of AC gain and we can see in figure 43 that it is near to 0 dB for low frequencies.

Common Source Stage

The common-source (CS) amplifier may be viewed as a voltage amplifier. Input voltage modulates the amount of current flowing through the transistor, changing the voltage across the output resistance according to Ohm's law, therefore changing output voltage, [14] .

This kind of amplifier stages can have a great gain even near to 30 dB.

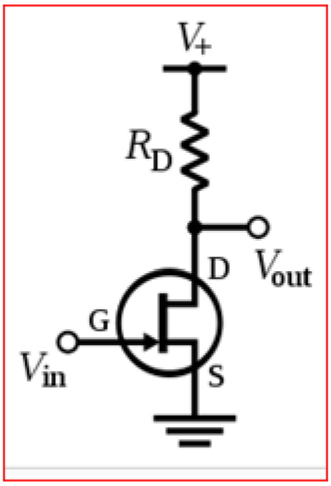

Figure 44: Common Source Stage model

We observe transistor drain AC voltage to obtain gain which in this case is slightly higher than 12 dB for low frequencies.

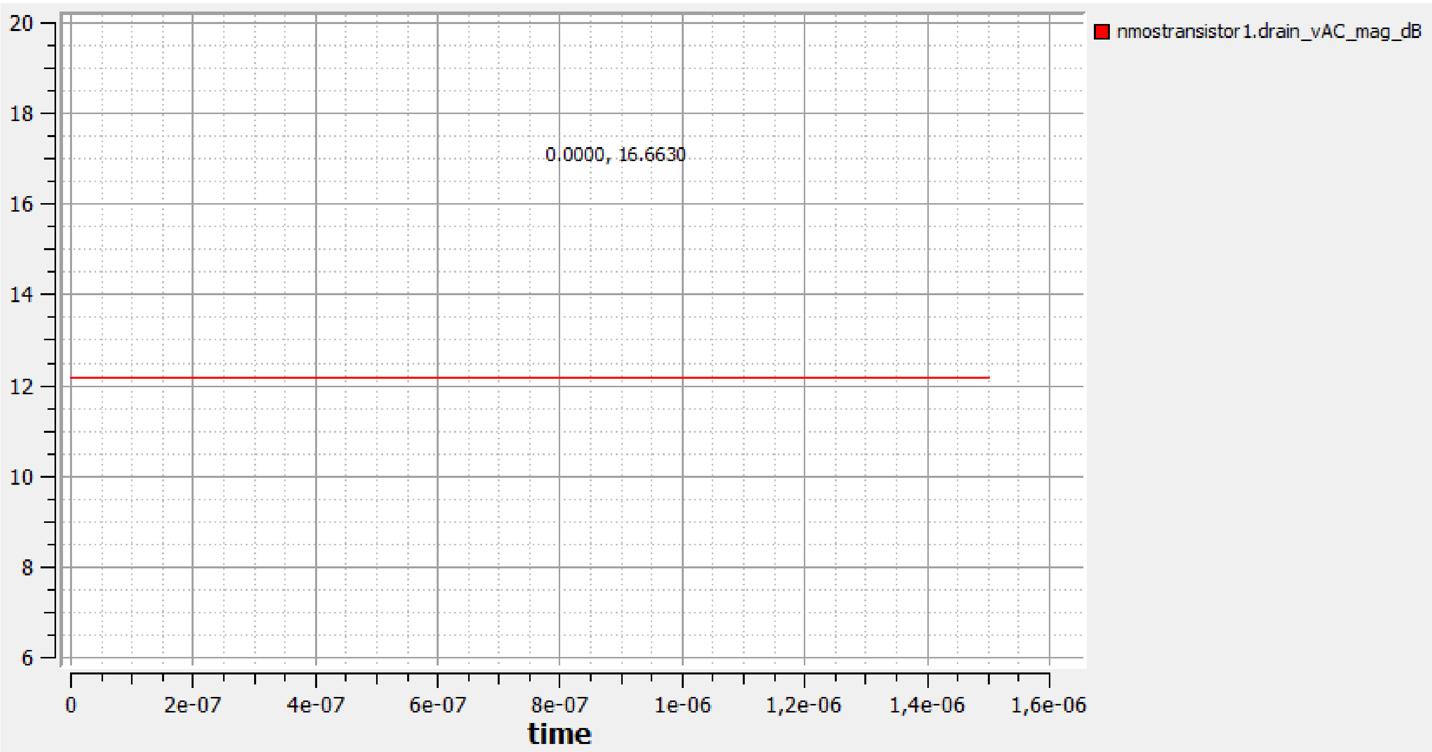

Figure 45: Common source stage AC plot

If we think of transient behavior, when input voltage raises, current through resistor rises and output voltage falls down. We take a look to transient variables to check they are consistent with this behavior.

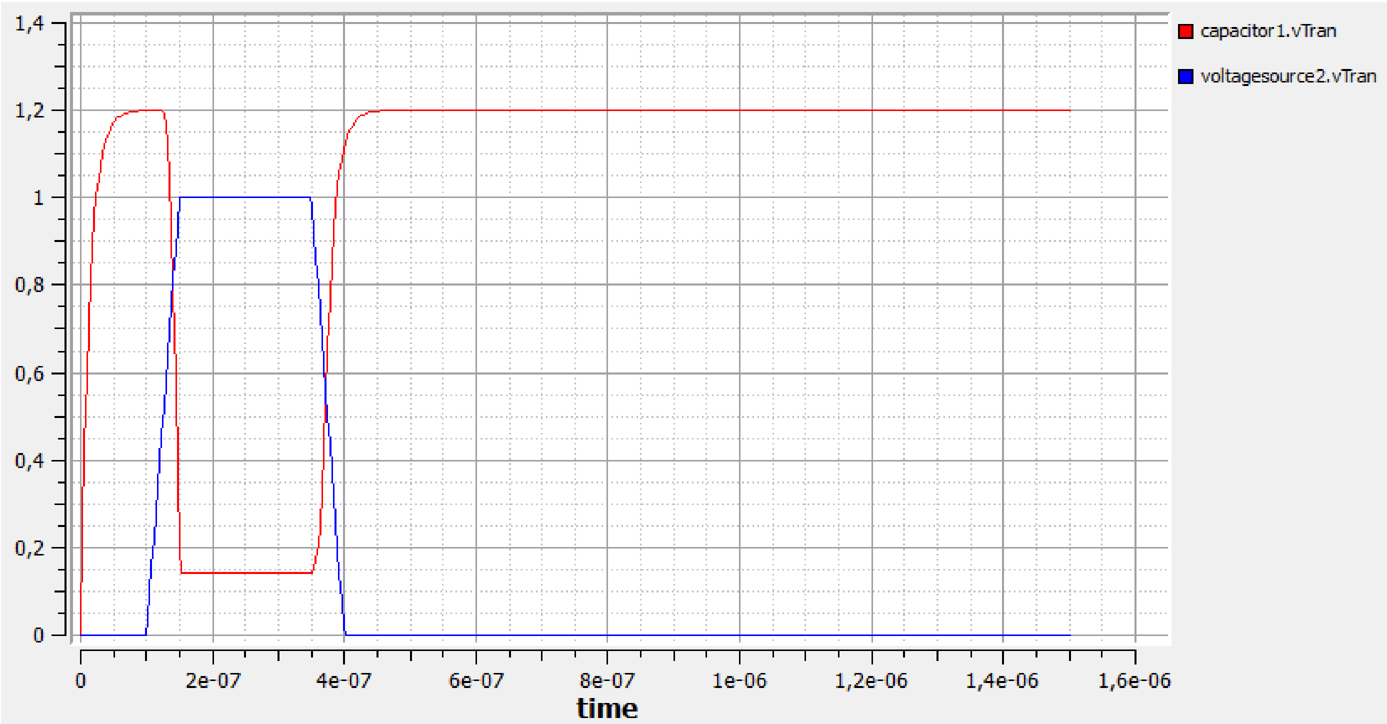

Figure 46: Common source stage transient plot

Current Mirror

A **current mirror** is a circuit designed to copy a current through one active device by controlling the current in another active device of a circuit, keeping the output current constant regardless of loading, [15]

In our case we copy current of current source which goes through transistor 2, to the current through transistor 1, see figure 26.

If we see DC currents of current source and transistor drain we can see they are really similar, as it should be.

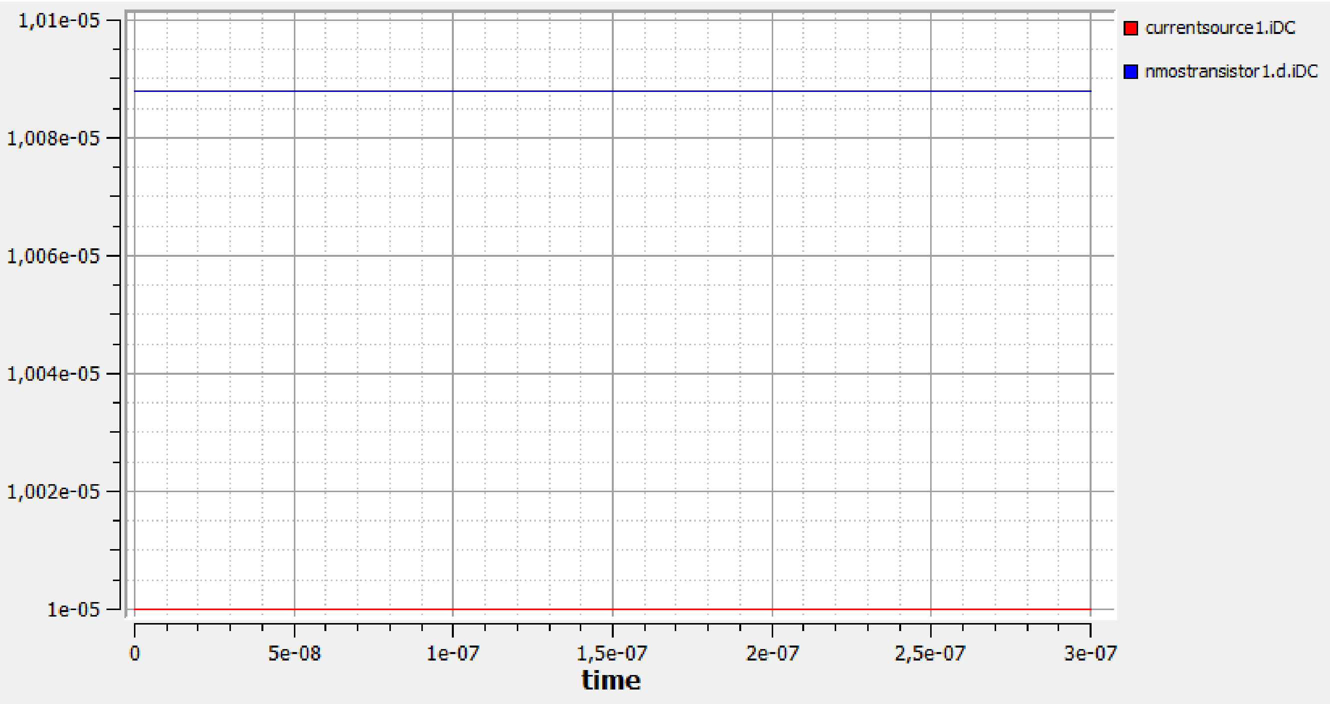

Figure 47: Current Mirror DC plot

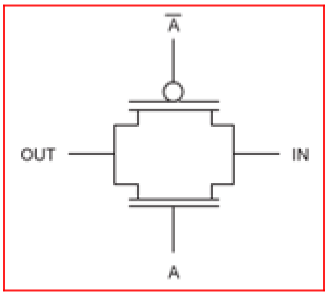

Figure 48: Transmission Gate model

Transmission Gate

A transmission gate, or analog switch, is defined as an electronic element that will selectively block or pass a signal level from the input to the output. This solid-state switch is comprised of
a PMOS transistor and NMOS transistor. The control gates are biased in a complementary manner so that both transistors are either on or off.

When the voltage on node A is a logical “1”, the complementary logical “0” is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a logical “0”, the complementary logical “1” is applied to node A, turning both transistors off and forcing a high-impedance condition in both the IN and OUT nodes, [16].
For testing it, we set a pulse in NMOS gate (A signal in the figure) and an opposite pulse in PMOS gate (to obtain opposite of A) and we observe input and output signals.

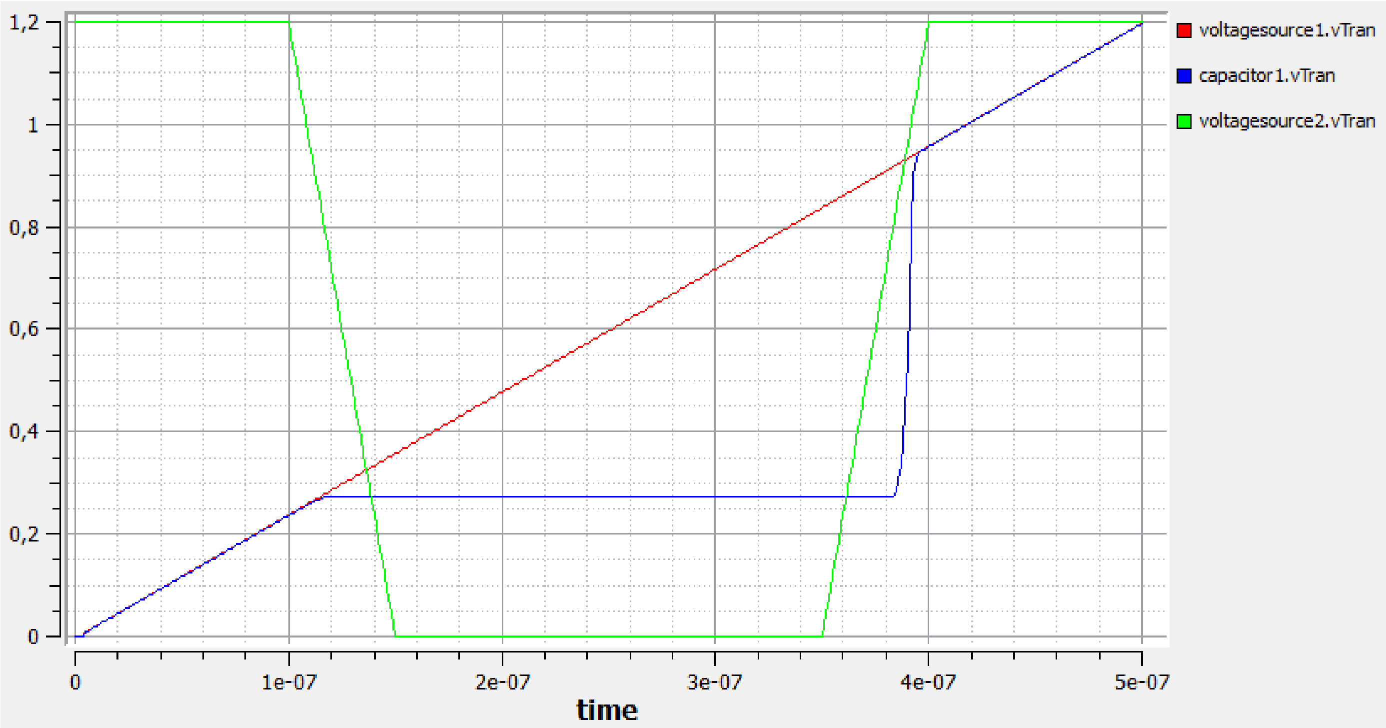

Figure 49: Transmission gate transient plot

As we expected when voltage in NMOS gate (voltagesource2) changes from logical 1 to logical 0, output signal does not follows input signal.

<u>Two Stages</u>

This circuit consists on just connecting the output of a common source amplifier to a common drain amplifier.

If we observe transient behavior we can see how transistor one gate (common drain transistor) acts as load for transistor two, and capacitor acts as load for transistor one.

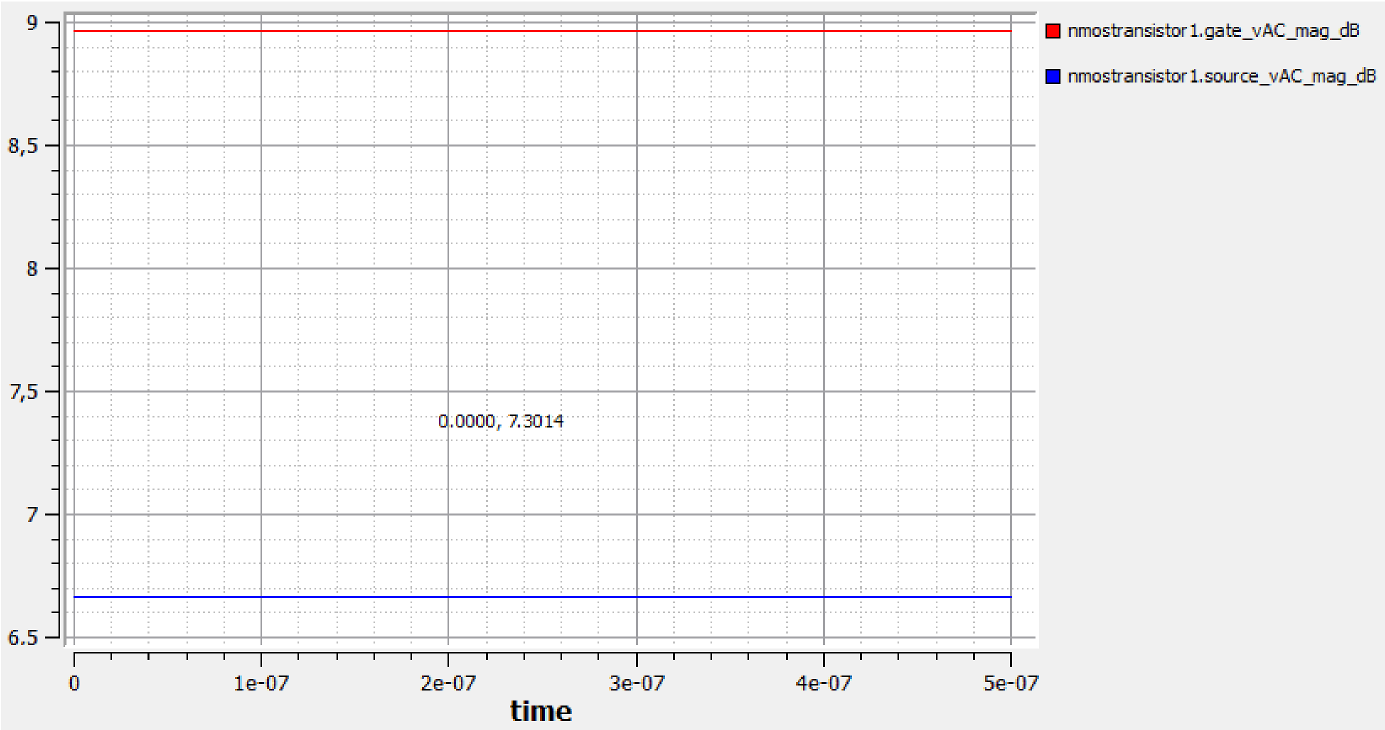

Figure 50: Two stages AC plot

And we can observe total gain (red line) is the gain of common source (blue line) plus common drain gain (more or less -2dB).

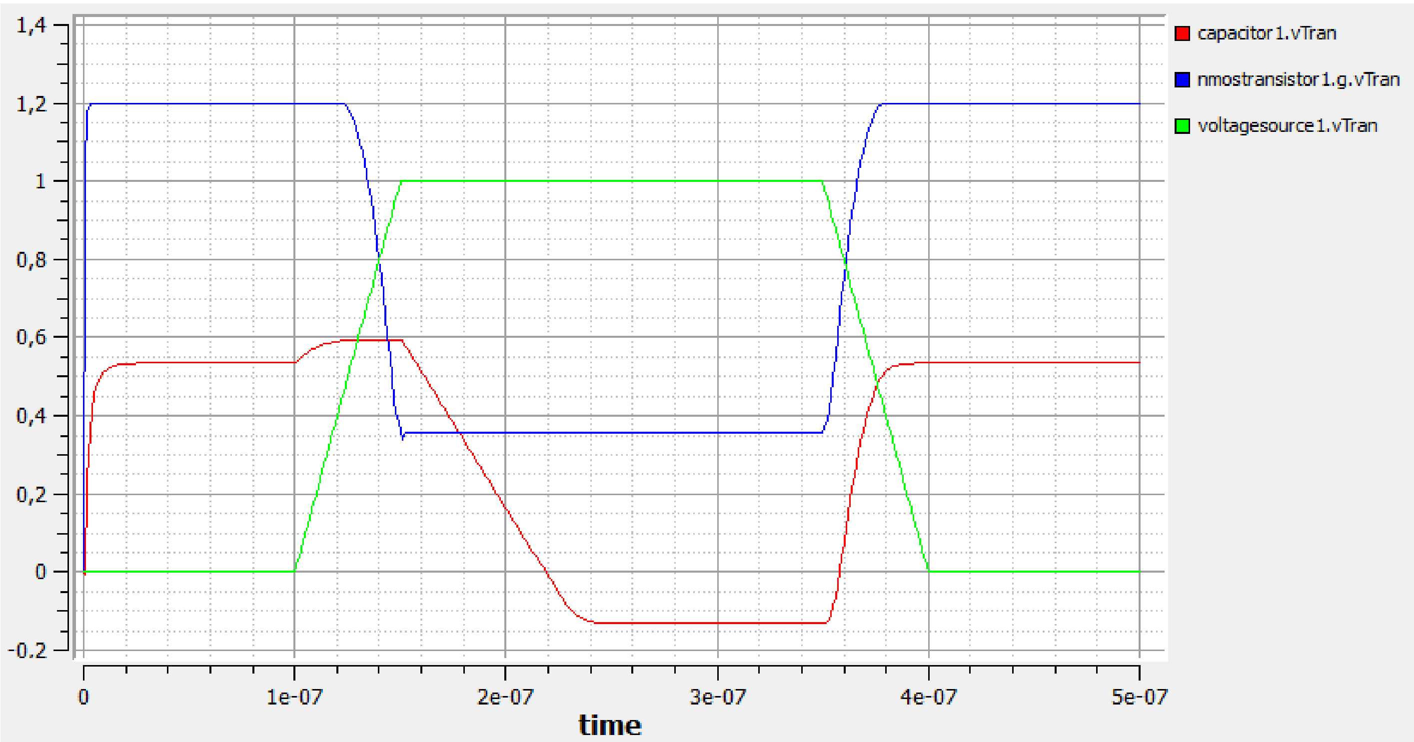

Figure 51: Two stages transient plot

6. Complementary Modules

In this section we will explain some complementary work done apart from library implementation. These complementary works consist of a python program which let you represent a variable of your model as a function of any other variable of the model instead of just time as it is done in OMEdit, and also a OMNotebook tutorial aimed for Ph.D. students who will work as lab assistants in labs where our library is used.

Python Program

The aim of this program is to implement what is known as a parametric sweep, that is, obtain the value of a variable as a function of another variable of our model and being able to plot it. The most common example is to plot some of the voltage of a pin as a function of the frequency.

To obtain this functionality we have modified a script which already exists made in [3].

What the scripts does is to simulate the model several times after changing the value of a parameter in each simulation. After each simulation it takes the values of the variables and writes them in a file so after several simulations we have the value of each variable for each simulation. After that, this file is given to Matlab where it is plotted.

Using this structure we have modified the script so our model is simulated with a different value of the parameter we choose each time. We keep the results after each simulation of the variable we are interested in and save it in a file following a format that is recognized by OMPlot (Plotting application of OpenModelica) and we launch it giving that file as a parameter.

It is recommended to set the number of simulations to a number lower than 10 so that it doesn’t takes more than a minute to make all the simulations.

Here we have an example of a result:

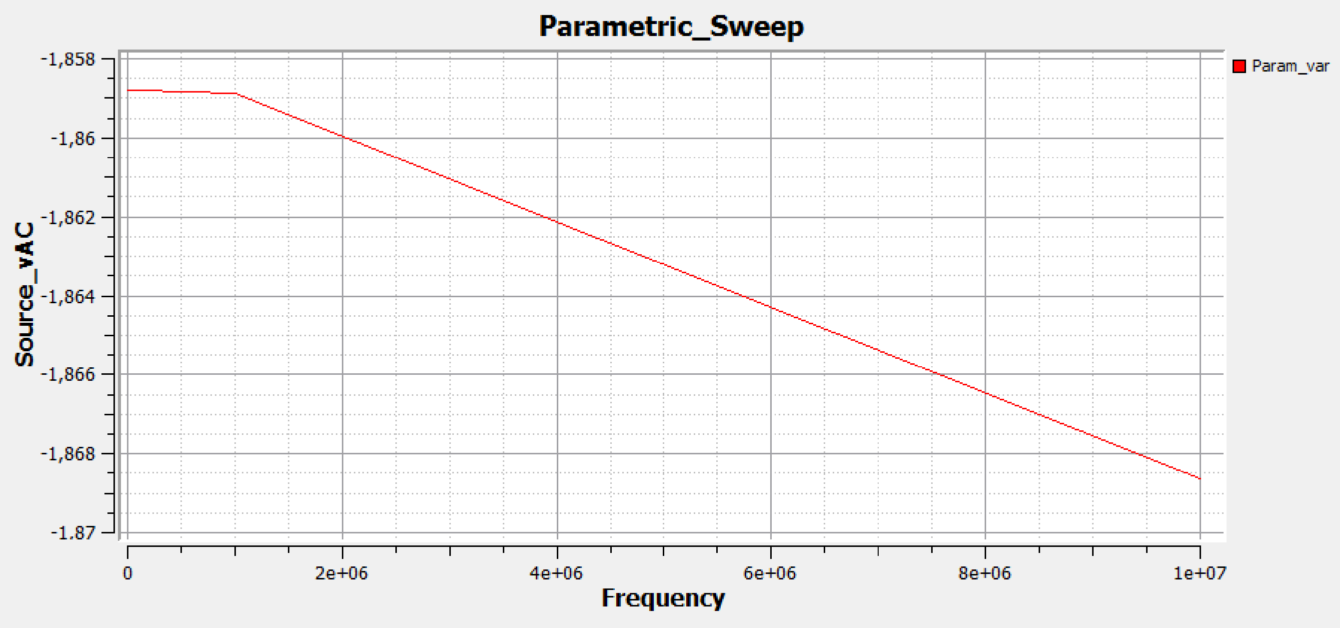

Figure 52: Parametric Sweep example

Code of the python program can be found in Appendix B.

OMNotebook Tutorial

Using OMNotebook functionality we have created a tutorial aimed for Ph.D. students which will participate in the creating of documentation and help to students in lab sessions.

This tutorial is divided in two sections.
First one, where it is explained how to create, simulate and plot variables of Common Source model which we use as example.
In second part, users are proposed to try creating by themselves the rest of examples sub-package models, providing also a solution to those exercises.

The only inconvenient of this tutorial is that models have to be created by code as if we were using OpenModelica console (OMShell), so it is also recommended to users try to do this exercises also using OMEdit since it is what students will usually use.

7. Conclusions and Future Works

Here we have described the devolpment of a library intended to simulation of integrated circuits. We have tried to make a simple but at the same time, useful library which could be used instead of expensive and complex simulation tools.

We have also try to keep a good level of complexity in our models but always having in mind the purpose of this project. Maybe this could be one of the principal points to improve in future works.

So we would recommend an improvement of the models to introduce some effects like temperature or noise to model equations giving them a more precise behavior.

Another task which would be interesting would be to adapt parametric sweep functionality given by python script to OpenModelica environment.

Also would be interesting to work on optimization of the models to improve simulation times which are maybe the weak point of this library. Related to that, another modification could be to make DC, AC and transient variables of all our components conditional to the value of three boolean parameters (one to each variable type), so when we make a parametric sweep, if we are only interested in an AC variable, we can “turn off” the other variables, reducing the time necessary to make all the simulations.

And finally, to start using this library as support to electronics lab it would be good to develop some kind of lab guide which let students to learn the basic principles of integrated circuits components and configurations using our library.

References

1 . Martin Otter and Hilding Elmqvist. Modelica Language, Libraries, Tools, Workshop and EU-Project RealSim, June 2001. https://www.modelica.org/documents/ModelicaOverview14.pdf, accessed August 2011.

2. SPICELib - www.modelica.org/libraries/SPICELib, accessed April 2011.

3. Mohsen Torabzadeh-Tari, Peter Fritzson, Martin Sjölund and Adrian Pop, 2009 - OpenModelica–Python Interoperability Applied to Monte Carlo Simulations – PELAB, Linköping University, Sweden.

4. Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica, Wiley-IEEE Press, January 2004.

5. OpenModelica – http://www.openmodelica.org, accessed December 2010.

6. Integrated Circuits - www.wikipedia.com, accessed March 2011.

7. caracteriz_cto_harris.pdf - https://sites.google.com/a/die.upm.es/lsi/People/marisa/teachingduties/mcre/mcre, accessed January 2011.

8. Iva Georgieva Marinova, Diana Ivanova Pukneva, Marin Hristov Hristov - The influence of geometry and temperature on the parameters of integrated capacitors - ECAD Laboratory, Technical University of Sofia, Bulgaria.

9. SPICE Level 1 MOSFET Model - http://www.csit-sun.pub.ro/courses/vlsi/VLSI_Darmstad/www.microelectronic.e-technik.tu-darmstadt.de/lectures/winter/vlsi/vorlesung_pdf/chap04.pdf - CSIT Laboratory, Politehnica University of Bucarest, accessed February 2011.

10. A. Urquía, C. Martín and S. Dormido - Design of SPICELib: A Modelica library for modeling and analysis of electric circuits - E.T.S. de Ingeniería Informática, U.N.E.D., Spain.

11. Robert Hägglund, K. Ola Andersson, Niklas U. Andersson, J. Jacob Wikner - Exercises manual (ATIK and ANIK courses).

12. CMOS - http://en.wikipedia.org/wiki/CMOS, accessed August 2011.

13. Common Drain - http://en.wikipedia.org/wiki/Common_drain, accessed August 2011.

14. Common Source - http://en.wikipedia.org/wiki/Common_source, accessed August 2011.

15. Current Mirror - http://en.wikipedia.org/wiki/Current_mirror, accessed August 2011.

16. Transmission Gate - http://www.maxim-ic.com/app-notes/index.mvp/id/4243, accessed August 2011.

Appendix A – Library Models

IC package

```
package IC "Library of components for integrated circuits"
  extends Modelica.Icons.Library2;
```

Components sub-package

```
package Components "Contains different components"
  extends Modelica.Icons.Library2;
  import SI = Modelica.SIunits;
  import nonSI = Modelica.SIunits.Conversions.NonSIunits;
  import INTERFACE = Modelica.Electrical.IC.Interfaces;
  import Modelica.Electrical.IC.Functions.*;


model Resistor "Ideal integrated electrical resistor"
  extends INTERFACE.OnePort;
  import Modelica.Electrical.IC.Constants.ResistorConstants.*;

  parameter SI.Distance W = 1 "Width of the resistor";
  parameter SI.Distance L = 20 "Length of the resistor";
  SI.Resistance R "Total resistance of the resistor";
  SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage across
    the component";
  SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
  across the component";
  nonSI.Angle_deg vAC_phase(start = 0) "Phase (deg) of AC small-signal
    voltage across the component";
  SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
  SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
    current";
  nonSI.Angle_deg iAC_phase(start = 0) "Phase (deg) of AC small-signal
    current";
  SI.Voltage pinP_vAC_mag(start = 0);
  nonSI.Angle_deg pinP_vAC_phase(start = 0);
  SI.Voltage pinP_vAC_mag_dB(start = 0);
  SI.Voltage pinN_vAC_mag(start = 0);
  nonSI.Angle_deg pinN_vAC_phase(start = 0);
  SI.Voltage pinN_vAC_mag_dB(start = 0);

equation
  R = Rs * L / W;
  assert(R > 50, "Resistance has a value lower than 50 Ohms, it is a very low
    value. Make sure that is the one you want");
  assert(R < 10000000, "Resistance has a value higher than 10 MOhms, it is a
    very high value. Make sure that is the one you want");
  vDC = R * iDC;
  vTran = R * iTran;
  {vAC_Re,vAC_Im} = R * {iAC_Re,iAC_Im};
  (pinP_vAC_mag,pinP_vAC_phase) = Rect2Polar({p.vAC_Re,p.vAC_Im});
  pinP_vAC_mag_dB = Decibels(pinP_vAC_mag);
  (pinN_vAC_mag,pinN_vAC_phase) = Rect2Polar({n.vAC_Re,n.vAC_Im});
  pinN_vAC_mag_dB = Decibels(pinN_vAC_mag);
  (vAC_mag,vAC_phase) = Rect2Polar({vAC_Re,vAC_Im});
  vAC_mag_dB = Decibels(vAC_mag);
  (iAC_mag,iAC_phase) = Rect2Polar({iAC_Re,iAC_Im});
```

```
  iAC_mag_dB = Decibels(iAC_mag);
end Resistor;

model Capacitor "Ideal integrated parallel-plate capacitor"
  extends INTERFACE.OnePort;
  outer SI.Frequency freq "AC small-signal analysis frequency";
  import Modelica.Electrical.IC.Constants.CapacitorConstants.*;

  parameter SI.Distance W = 4e-06 "Width of the capacitor";
  parameter SI.Distance L = 4e-06 "Length of the capacitor";
  SI.Capacitance C "Capacitance of the capacitor";
  SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage across
    the component";
  SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
  across the component";
  nonSI.Angle_deg vAC_phase(start = 0) "Phase (deg) of AC small-signal
    voltage across the component";
  SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
  SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
   current";
  nonSI.Angle_deg iAC_phase(start = 0) "Phase (deg) of AC small-signal
  current";
  SI.Voltage pinP_vAC_mag(start = 0);
  nonSI.Angle_deg pinP_vAC_phase(start = 0);
  SI.Voltage pinP_vAC_mag_dB(start = 0);
  SI.Voltage pinN_vAC_mag(start = 0);
  nonSI.Angle_deg pinN_vAC_phase(start = 0);
  SI.Voltage pinN_vAC_mag_dB(start = 0);

equation
  C = (0.0885 * W * L * Er * 1000000000000.0) / (t * 10000000000.0) * 1e-12;
  assert(C > 1e-15, "Capacitor has a value lower than 1 femtoFaradio, it is a
    very low value. Make sure that is the one you want");
  assert(C < 1e-09, "Capacitor has a value higher than 1 nanoFaradio, it is a
    very high value. Make sure that is the one you want");
  iDC = 0;
  iTran = C * der(vTran);
  {vAC_Re,vAC_Im} * 2 * pi * freq * C = {iAC_Im,-iAC_Re};
  (pinP_vAC_mag,pinP_vAC_phase) = Rect2Polar({p.vAC_Re,p.vAC_Im});
  pinP_vAC_mag_dB = Decibels(pinP_vAC_mag);
  (pinN_vAC_mag,pinN_vAC_phase) = Rect2Polar({n.vAC_Re,n.vAC_Im});
  pinN_vAC_mag_dB = Decibels(pinN_vAC_mag);
  (vAC_mag,vAC_phase) = Rect2Polar({vAC_Re,vAC_Im});
  vAC_mag_dB = Decibels(vAC_mag);
  (iAC_mag,iAC_phase) = Rect2Polar({iAC_Re,iAC_Im});
  iAC_mag_dB = Decibels(iAC_mag);
end Capacitor;


model Inductor "Ideal inductor"
  extends INTERFACE.OnePort;
  outer SI.Frequency freq "AC small-signal analysis frequency";
  import Modelica.Electrical.IC.Constants.InductorConstants.*;

  parameter SI.Inductance L = 5e-08 "Inductance of the inductor";
  SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage across
  the inductor";
  SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
  across the inductor";
  nonSI.Angle_deg vAC_phase(start = 0) "Phase (deg) of AC small-signal
    voltage across the inductor";
  SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
  SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
    current";
  nonSI.Angle_deg iAC_phase(start = 0) "Phase (deg) of AC small-signal
    current";
  SI.Voltage pinP_vAC_mag(start = 0);
  SI.Voltage pinP_vAC_phase(start = 0);
```

```
  SI.Voltage pinP_vAC_mag_dB(start = 0);
  SI.Voltage pinN_vAC_mag(start = 0);
  SI.Voltage pinN_vAC_phase(start = 0);
  SI.Voltage pinN_vAC_mag_dB(start = 0);

equation
  vDC = 0;
  vTran = L * der(iTran);
  {vAC_Re,vAC_Im} = {iAC_Im,-iAC_Re} * 2 * pi * freq * L;
  (pinP_vAC_mag,pinP_vAC_phase) = Rect2Polar({p.vAC_Re,p.vAC_Im});
  pinP_vAC_mag_dB = Decibels(pinP_vAC_mag);
  (pinN_vAC_mag,pinN_vAC_phase) = Rect2Polar({n.vAC_Re,n.vAC_Im});
  pinN_vAC_mag_dB = Decibels(pinN_vAC_mag);
  (vAC_mag,vAC_phase) = Rect2Polar({vAC_Re,vAC_Im});
  vAC_mag_dB = Decibels(vAC_mag);
  (iAC_mag,iAC_phase) = Rect2Polar({iAC_Re,iAC_Im});
  iAC_mag_dB = Decibels(iAC_mag);
end Inductor;


model Ground
  INTERFACE.PositivePin p;

equation
  {p.vDC,p.vTran,p.vAC_Re,p.vAC_Im} = zeros(4);
end Ground;


model NMOStransistor "Spice Level 1 NMOS transistor model"
  extends INTERFACE.MOSFET;
  import
  Modelica.Electrical.IC.Constants.NMOStransistorConstants.*;

  inner SI.Voltage vdsDC "Drain to source voltage";
  inner SI.Voltage vgsDC "Gate to source voltage";
  inner SI.Voltage vbsDC "Bulk to source voltage";
  inner SI.Voltage vdsTran "Drain to source voltage";
  inner SI.Voltage vgsTran "Gate to source voltage";
  inner SI.Voltage vbsTran "Bulk to source voltage";
  SI.Voltage drain_vAC_mag "Magnitude of AC small-signal drain voltage";
  SI.Voltage drain_vAC_mag_dB "Magnitude (dB) of AC small-signal drain
    voltage";
  nonSI.Angle_deg drain_vAC_phase(start = 0) "Phase (deg) of AC small-signal
    drain voltage";
  SI.Voltage source_vAC_mag "Magnitude of AC small-signal source voltage";
  SI.Voltage source_vAC_mag_dB "Magnitude (dB) of AC small-signal source
    voltage";
  nonSI.Angle_deg source_vAC_phase(start = 0) "Phase (deg) of drain AC small-
    signal source voltage";
  SI.Voltage bulk_vAC_mag "Magnitude of AC small-signal bulk voltage";
  SI.Voltage bulk_vAC_mag_dB "Magnitude (dB) of AC small-signal bulk voltage";
  nonSI.Angle_deg bulk_vAC_phase(start = 0) "Phase (deg) of AC small-signal
    bulk voltage";
  SI.Voltage gate_vAC_mag "Magnitude of AC small-signal gate voltage";
  SI.Voltage gate_vAC_mag_dB "Magnitude (dB) of AC small-signal gate voltage";
  nonSI.Angle_deg gate_vAC_phase(start = 0) "Phase (deg) of AC small-signal
    gate voltage";
  SI.Current drain_iAC_mag "Magnitude of AC small-signal drain current";
  SI.Current drain_iAC_mag_dB "Magnitude (dB) of AC small-signal drain
    current";
  nonSI.Angle_deg drain_iAC_phase(start = 0) "Phase (deg) of AC small-signal
    drain current";
  SI.Current source_iAC_mag "Magnitude of AC small-signal source current";
  SI.Current source_iAC_mag_dB "Magnitude (dB) of AC small-signal source
    current";
  nonSI.Angle_deg source_iAC_phase(start = 0) "Phase (deg) of drain AC small-
    signal source current";
  SI.Current bulk_iAC_mag "Magnitude of AC small-signal bulk current";
```

```
  SI.Current bulk_iAC_mag_dB "Magnitude (dB) of AC small-signal bulk current";
  nonSI.Angle_deg bulk_iAC_phase(start = 0) "Phase (deg) of AC small-signal
    bulk current";
  SI.Current gate_iAC_mag "Magnitude of AC small-signal gate current";
  SI.Current gate_iAC_mag_dB "Magnitude (dB) of AC small-signal gate current";
  nonSI.Angle_deg gate_iAC_phase(start = 0) "Phase (deg) of AC small-signal
    gate current";
  parameter SI.Length L = 1e-07 "Gate length";
  parameter SI.Length W = 4e-06 "Gate width";
  inner SI.Voltage vdsDCSgn "Drain-pin to source-pin voltage";
  inner SI.Voltage vthDC "Threshold voltage";
  inner SI.Voltage vdsTranSgn "Drain-pin to source-pin voltage";
  inner SI.Voltage vthTran "Threshold voltage";
  inner SI.Voltage gate_vAC_Re;
  inner SI.Voltage gate_vAC_Im;
  inner SI.Voltage bulk_vAC_Re;
  inner SI.Voltage bulk_vAC_Im;

  NMOS.Ids Ids1(VTO = VTO, GAMMA = GAMMA, PHI = PHI, KP = KP, W = W, L = L,
    LD = LD, LAMBDA = LAMBDA);
  NMOS.Idiode Dbs(IS = IS);
  NMOS.Cdiode Cbs(CJ = CJ, CJSW = CJSW, MJ = MJ, MJSW = MJSW, FC = FC, PB =
    PB, P = PS, A = AS;
  NMOS.Idiode Dbd(IS = IS);
  NMOS.Cdiode Cbd(CJ = CJ, CJSW = CJSW, MJ = MJ, MJSW = MJSW, FC = FC, PB =
    PB, P = PD, A = AD);
  NMOS.Cgd Cgd1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
    CGDO = CGDO, CGSO = CGSO, gateSourcePinsC = false);
  NMOS.Cgd Cgs1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
    CGDO = CGDO, CGSO = CGSO, gateSourcePinsC = true);
  NMOS.Cgb Cgb1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
    CGBO = CGBO);

equation
  vthDC = VTO + GAMMA * (sqrt(abs(PHI - vbsDC)) - sqrt(PHI));
  vthTran = VTO + GAMMA * (sqrt(abs(PHI - vbsTran)) - sqrt(PHI));
  vdsDC = noEvent(abs(Ids1.vDC));
  vdsDCSgn = Ids1.vDC;
  vdsTran = noEvent(abs(Cgs1.vTran - Cgd1.vTran));
  vdsTranSgn = Cgs1.vTran - Cgd1.vTran;
  vgsDC = max({Cgs1.vDC,Cgd1.vDC});
  vgsTran = max({Cgs1.vTran,Cgd1.vTran});
  vbsDC = max({Cbs.vDC,Cbd.vDC});
  vbsTran = max({Cbs.vTran,Cbd.vTran});
  gate_vAC_Re = g.vAC_Re;
  gate_vAC_Im = g.vAC_Im;
  bulk_vAC_Re = b.vAC_Re;
  bulk_vAC_Im = b.vAC_Im;
  (drain_vAC_mag,drain_vAC_phase) = Rect2Polar({d.vAC_Re,d.vAC_Im});
  drain_vAC_mag_dB = Decibels(drain_vAC_mag);
  (drain_iAC_mag,drain_iAC_phase) = Rect2Polar({d.iAC_Re,d.iAC_Im});
  drain_iAC_mag_dB = Decibels(drain_iAC_mag);
  (source_vAC_mag,source_vAC_phase) = Rect2Polar({s.vAC_Re,s.vAC_Im});
  source_vAC_mag_dB = Decibels(source_vAC_mag);
  (source_iAC_mag,source_iAC_phase) = Rect2Polar({s.iAC_Re,s.iAC_Im});
  source_iAC_mag_dB = Decibels(source_iAC_mag);
  (bulk_vAC_mag,bulk_vAC_phase) = Rect2Polar({b.vAC_Re,b.vAC_Im});
  bulk_vAC_mag_dB = Decibels(bulk_vAC_mag);
  (bulk_iAC_mag,bulk_iAC_phase) = Rect2Polar({b.iAC_Re,b.iAC_Im});
  bulk_iAC_mag_dB = Decibels(bulk_iAC_mag);
  (gate_vAC_mag,gate_vAC_phase) = Rect2Polar({g.vAC_Re,g.vAC_Im});
  gate_vAC_mag_dB = Decibels(gate_vAC_mag);
  (gate_iAC_mag,gate_iAC_phase) = Rect2Polar({g.iAC_Re,g.iAC_Im});
  gate_iAC_mag_dB = Decibels(gate_iAC_mag);
  connect(d,Ids1.p);
  connect(Ids1.n,s);
  connect(g,Cgd1.p);
  connect(g,Cgs1.p);
```

```
  connect(g,Cgb1.p);
  connect(b,Dbs.p);
  connect(b,Cbs.p);
  connect(b,Dbd.p);
  connect(b,Cbd.p);
  connect(b,Cgb1.n;
  connect(Ids1.n,Cgs1.n);
  connect(Ids1.p,Cgd1.n);
  connect(Ids1.n,Cbs.n);
  connect(Ids1.p,Cbd.n);
  connect(Cbd.n,Dbd.n);
  connect(Dbs.n,Cbs.n);
end NMOStransistor;


package NMOS "Contains the components used to model an NMOS transistor"
  extends Modelica.Icons.Library2;

model Idiode
  extends INTERFACE.OnePort;
  parameter SI.Current IS "Reverse saturation current at 300K";
  constant SI.Voltage thermalVolt = 300 / 11600 "Thermal voltage";
  constant SI.Conductance GMIN = 1e-12 "Conductance in parallel with the pn-
  junction";
  SI.Conductance gAC(start = 1) "AC small-signal conductance";

equation
  iDC = IS * (exp(vDC / thermalVolt) - 1) + vDC * GMIN;
  iTran = if noEvent(vTran > 0) then IS * (exp(vTran / thermalVolt) - 1) +
  vTran * GMIN else (IS / thermalVolt + GMIN) * vTran;
  gAC = if vDC > 0 then IS / thermalVolt * exp(vDC / thermalVolt) + GMIN else
  IS / thermalVolt + GMIN;
  {iAC_Re,iAC_Im} = gAC * {vAC_Re,vAC_Im};
end Idiode;


model Ids
  extends INTERFACE.OnePort;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vgsDC "Gate to source voltage";
  outer SI.Voltage vbsDC "Bulk to source voltage";
  outer SI.Voltage vdsDC "Drain to source voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vgsTran "Gate to source voltage";
  outer SI.Voltage vbsTran "Bulk to source voltage";
  outer SI.Voltage vdsTran "Drain to source voltage";
  outer SI.Voltage vdsTranSgn "Drain-pin to source-pin voltage";
  outer SI.Voltage gate_vAC_Re;
  outer SI.Voltage gate_vAC_Im;
  outer SI.Voltage bulk_vAC_Re;
  outer SI.Voltage bulk_vAC_Im;
  parameter SI.Voltage VTO "Zero-bias threshold voltage";
  parameter Real GAMMA "Body-effect parameter [V0.5]";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter Real KP "Transconductance parameter [A/V2]";
  parameter SI.Length W "Gate width";
  parameter SI.Length L "Gate length";
  parameter SI.Length LD "Lateral diffusion";
  parameter Real LAMBDA "Channel-length modulation [V-1]";
protected
  parameter Real beta = (KP * W) / (L - 2 * LD);
  Boolean pinNisSourceAC(start = true);
  SI.Current idsDC "Drain to source current";
  SI.Current idsTran "Drain to source current";
  Real ARG(start = 0);
  SI.Conductance gdsAC "dIds/dVds";
  SI.Conductance gmbsAC "dIds/dVbs";
  SI.Conductance gmAC "dIds/dVgs";
```

```
  SI.Voltage vdsAC_Re;
  SI.Voltage vdsAC_Im;
  SI.Voltage vgsAC_Re;
  SI.Voltage vgsAC_Im;
  SI.Voltage vbsAC_Re;
  SI.Voltage vbsAC_Im;
  SI.Current idsAC_Re;
  SI.Current idsAC_Im;

equation
  iDC = noEvent(if vDC > 0 then idsDC else -idsDC);
  idsDC = if vgsDC <= vthDC then 0 else noEvent(if vdsDC < vgsDC - vthDC then
  beta * (vgsDC - vthDC - 0.5 * vdsDC) * vdsDC * (1 + LAMBDA * vdsDC) else
    0.5 * beta * (vgsDC - vthDC) ^ 2 * (1 + LAMBDA * vdsDC));
  iTran = if noEvent(vdsTranSgn >= 0) then idsTran else -idsTran;
  idsTran = if vgsTran <= vthTran then 0 else if noEvent(vdsTran < vgsTran -
    vthTran) then beta * (vgsTran - vthTran - 0.5 * vdsTran) * vdsTran * (1 +
    LAMBDA * vdsTran) else 0.5 * beta * (vgsTran - vthTran) ^ 2 * (1 + LAMBDA
    * vdsTran);
  {vdsAC_Re,vdsAC_Im} = if pinNisSourceAC then {vAC_Re,vAC_Im} else -
    {vAC_Re,vAC_Im};
  {vgsAC_Re,vgsAC_Im} = if pinNisSourceAC then {gate_vAC_Re -
    n.vAC_Re,gate_vAC_Im - n.vAC_Im} else {gate_vAC_Re - p.vAC_Re,gate_vAC_Im
    - p.vAC_Im};
  {vbsAC_Re,vbsAC_Im} = if pinNisSourceAC then {bulk_vAC_Re -
    n.vAC_Re,bulk_vAC_Im - n.vAC_Im} else {bulk_vAC_Re - p.vAC_Re,bulk_vAC_Im
    - p.vAC_Im};
  {idsAC_Re,idsAC_Im} = gdsAC * {vdsAC_Re,vdsAC_Im} + gmbsAC *
    {vbsAC_Re,vbsAC_Im} + gmAC * {vgsAC_Re,vgsAC_Im};
    {iAC_Re,iAC_Im} = if pinNisSourceAC then {idsAC_Re,idsAC_Im} else -
    {idsAC_Re,idsAC_Im};
  pinNisSourceAC = vDC >= 0;
  gdsAC = if vgsDC <= vthDC then 0 else if vdsDC < vgsDC - vthDC then beta *
    (vgsDC - vthDC - vdsDC) * (1 + LAMBDA * vdsDC) + beta * (vgsDC - vthDC -
    0.5 * vdsDC) * LAMBDA * vdsDC else 0.5 * beta * (vgsDC - vthDC) ^ 2 *
    LAMBDA;
  gmbsAC = gmAC * ARG;
  ARG = if vbsDC > 0 then (0.5 * GAMMA) / (sqrt(PHI) - (0.5 * vbsDC) /
    sqrt(PHI)) else (0.5 * GAMMA) / sqrt(PHI - vbsDC);
  gmAC = if vgsDC <= vthDC then 0 else if vdsDC < vgsDC - vthDC then beta *
    vdsDC * (1 + LAMBDA * vdsDC) else beta * (vgsDC - vthDC) * (1 + LAMBDA *
    vdsDC);
end Ids;


partial model Capacitor
  extends INTERFACE.OnePort;
  outer SI.Frequency freq "Frequency of the source";
  constant Real pi = 3.14159265358979;
  SI.Capacitance Cvar(start = 1e-12) "Transient analysis capacitance";
  SI.Capacitance CvarAC(start = 1e-12) "AC small-signal capacitance";

equation
  iDC = 0;
  {vAC_Re,vAC_Im} * 2 * pi * freq * CvarAC = {iAC_Im,-iAC_Re};
end Capacitor;


model Cdiode
  extends Capacitor;
  parameter Real CJ "Capacitance at zero-bias voltage per squere meter of
    area [F/m2]";
  parameter Real CJSW "Capacitance at zero-bias voltage per meter of
    perimeter [F/m]";
  parameter Real MJ "Bulk junction capacitnce grading coefficient";
  parameter Real MJSW "Perimeter capacitance grading coefficient";
  parameter Real FC "Substrate-junction forward-bias coefficient";
  parameter SI.Voltage PB "Junction potencial";
```

```
  parameter SI.Length P "Junction perimeter";
  parameter SI.Area A "Junction area";
protected
  parameter Real F2 = (1 - FC) ^ (1 + MJ);
  parameter Real F3 = 1 - FC * (1 + MJ);
  parameter Real FCtimesPB = FC * PB;
  parameter Real CJtimesA = CJ * A;
  parameter Real CJSWtimesP = CJSW * P;
  parameter Real coef1 = F3 * (CJtimesA / F2 + (CJSW * P) / F2);
  parameter Real coef2 = ((CJtimesA * MJ) / F2 + (CJSW * P * MJSW) / F2) / PB;
  parameter Real invPB = 1 / PB;

 equation
  Cvar = if noEvent(vTran < FCtimesPB) then CJtimesA / (1 - invPB * vTran) ^
    MJ + CJSWtimesP / (1 - invPB * vTran) ^ MJSW else coef1 + coef2 * vTran;
  Cvar * der(vTran) = iTran;
  CvarAC = if vDC < FCtimesPB then CJtimesA / (1 - invPB * vDC) ^ MJ +
    CJSWtimesP / (1 - invPB * vDC) ^ MJSW else coef1 + coef2 * vDC;
 end Cdiode;


 model Cgb
  extends Capacitor;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vdsDC "Drain to source voltage";
  outer SI.Voltage vgsDC "Gate to source voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vdsTran "Drain to source voltage";
  outer SI.Voltage vgsTran "Gate to source voltage";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter SI.Length LD "Lateral diffusion";
  parameter SI.Length W "Gate width";
  parameter SI.Length L "Gate length";
  parameter SI.Length TOX "Gate oxide thickness";
  parameter Real EPSR "Dielectric constant of the oxide";
  parameter Real CGBO "Gate-bulk overlap capacitance per meter [F/m]";
protected
  parameter SI.Length LEFF = L + 2 * LD "Effective length";
  parameter SI.Capacitance COX = (EPS0 * EPSR * W * LEFF) / TOX "Gate oxide
  capacitance";
  parameter Real CGBOtimesLEFF = CGBO * LEFF;
  parameter Real COXtimesinvPHI = COX / PHI;
  constant Real EPS0 = 8.85e-12 "Permittivity of free space [F/m]";

 equation
  Cvar - CGBOtimesLEFF = if noEvent(vgsTran < vthTran - PHI) then COX else if
    noEvent(vgsTran < vthTran) then COXtimesinvPHI * (vthTran - vgsTran) else
    0;
  Cvar * der(vTran) = iTran;
  CvarAC = if vgsDC < vthDC - PHI then COX + CGBOtimesLEFF else if vgsDC <
    vthDC then COXtimesinvPHI * (vthDC - vgsDC) + CGBOtimesLEFF else
    CGBOtimesLEFF;
 end Cgb;


 model Cgd
  extends Capacitor;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vdsDC "Drain to source voltage";
  outer SI.Voltage vdsDCSgn "Drain-pin to source-pin voltage";
  outer SI.Voltage vgsDC "Gate to source voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vdsTran "Drain to source voltage";
  outer SI.Voltage vdsTranSgn "Drain-pin to source-pin voltage";
  outer SI.Voltage vgsTran "Gate to source voltage";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter SI.Length LD "Lateral diffusion";
  parameter SI.Length W "Gate width";
```

```
  parameter SI.Length L "Gate length";
  parameter SI.Length TOX "Gate oxide thickness";
  parameter Real EPSR "Dielectric constant of the oxide";
  parameter Real CGDO "Gate-drain overlap capacitance per meter [F/m]";
  parameter Real CGSO "Gate-source overlap capacitance per meter [F/m]";
  parameter Boolean gateSourcePinsC;
protected
  parameter SI.Length LEFF = L + 2 * LD "Effective length";
  parameter SI.Capacitance COX = (EPS0 * EPSR * W * LEFF) / TOX "Gate oxide
    capacitance";
  constant Real EPS0 = 8.85e-12 "Permittivity of free space [F/m]";
  parameter Real CGSOtimesW = CGSO * W;
  parameter Real CGDOtimesW = CGDO * W;
  parameter Real twoThirdsCOX = 2 / 3 * COX;
  parameter Real threeFourthsCOX = (3 * COX) / 4;
  parameter Real halfCOX = 0.5 * COX;
  parameter SI.Voltage vdsTranEPS = 0.0001;
  parameter SI.Voltage vdsDCEPS = 0.0001;
  Real Cgs;
  Real Cgd;
  Real CgsAC;
  Real CgdAC;
  Real CpinsGS;
  Real CpinsGD;
  Real CpinsGSAC;
  Real CpinsGDAC;

 equation
  Cvar * der(vTran) = iTran;
  Cvar = if gateSourcePinsC then CpinsGS else CpinsGD;
  CpinsGS = if noEvent(vdsTranSgn < -vdsTranEPS) then Cgd else if
    noEvent(vdsTranSgn < vdsTranEPS) then (0.5 * (Cgs - Cgd) * vdsTranSgn) /
    vdsTranEPS + 0.5 * (Cgs + Cgd) else Cgs;
  CpinsGD = if noEvent(vdsTranSgn < -vdsTranEPS) then Cgs else if
    noEvent(vdsTranSgn < vdsTranEPS) then (0.5 * (Cgd - Cgs) * vdsTranSgn) /
    vdsTranEPS + 0.5 * (Cgs + Cgd) else Cgd;
  Cgs - CGSOtimesW = if noEvent(vgsTran <= vthTran - PHI) then 0 else if
    noEvent(vgsTran <= vthTran) then twoThirdsCOX * ((-vthTran + vgsTran) /
    PHI + 1) else if noEvent(vgsTran <= vthTran + vdsTran) then twoThirdsCOX
    else if noEvent(vdsTran <= 2 * vdsTranEPS) then halfCOX else twoThirdsCOX
    * (1 - ((vgsTran - vdsTran - vthTran) / (2 * (vgsTran - vthTran) -
    vdsTran)) ^ 2);
  Cgd - CGDOtimesW = if noEvent(vgsTran <= vthTran + vdsTran) then 0 else if
    noEvent(vdsTran <= 2 * vdsTranEPS) then threeFourthsCOX else COX * (1 -
    ((vgsTran - vthTran) / (2 * (vgsTran - vthTran) - vdsTran)) ^ 2);
  CvarAC = if gateSourcePinsC then CpinsGSAC else CpinsGDAC;
  CpinsGSAC = if noEvent(vdsDCSgn < -vdsDCEPS) then CgdAC else if
    noEvent(vdsDCSgn < vdsDCEPS) then (0.5 * (CgsAC - CgdAC) * vdsDCSgn) /
    vdsDCEPS + 0.5 * (CgsAC + CgdAC) else CgsAC;
  CpinsGDAC = if noEvent(vdsDCSgn < -vdsDCEPS) then CgsAC else if
    noEvent(vdsDCSgn < vdsDCEPS) then (0.5 * (CgdAC - CgsAC) * vdsDCSgn) /
    vdsDCEPS + 0.5 * (CgsAC + CgdAC) else CgdAC;
  CgsAC = if vgsDC <= vthDC - PHI then CGSOtimesW else if vgsDC <= vthDC then
    twoThirdsCOX * ((-vthDC + vgsDC) / PHI + 1) + CGSOtimesW else if vgsDC <=
    vthDC + vdsDC then twoThirdsCOX + CGSOtimesW else if vdsDC <= 2 *
    vdsDCEPS   then halfCOX + CGSOtimesW else twoThirdsCOX * (1 - ((vgsDC -
    vdsDC - vthDC) / (2 * (vgsDC - vthDC) - vdsDC)) ^ 2) + CGSOtimesW;
  CgdAC = if vgsDC <= vthDC + vdsDC then CGDOtimesW else if vdsDC <= 2 *
    vdsDCEPS then threeFourthsCOX + CGDOtimesW else COX * (1 - ((vgsDC -
    vthDC) / (2 * (vgsDC - vthDC) - vdsDC)) ^ 2) + CGDOtimesW;
 end Cgd;
 end NMOS;


 model PMOStransistor "Spice Level 1 PMOS transistor model"
  extends INTERFACE.MOSFET;
  import
  Modelica.Electrical.IC.Constants.PMOStransistorConstants.*;
```

```
 inner SI.Voltage vsdDC "Source to drain voltage";
 inner SI.Voltage vsgDC "Source to gate voltage";
 inner SI.Voltage vsbDC "Source to bulk voltage";
 inner SI.Voltage vsdTran "Source to drain voltage";
 inner SI.Voltage vsgTran "Source to gate voltage";
 inner SI.Voltage vsbTran "Source to bulk voltage";
 SI.Voltage drain_vAC_mag "Magnitude of AC small-signal drain voltage";
 SI.Voltage drain_vAC_mag_dB "Magnitude (dB) of AC small-signal drain
   voltage";
 nonSI.Angle_deg drain_vAC_phase(start = 0) "Phase (deg) of AC small-signal
   drain voltage";
 SI.Voltage source_vAC_mag "Magnitude of AC small-signal source voltage";
 SI.Voltage source_vAC_mag_dB "Magnitude (dB) of AC small-signal source
   voltage";
nonSI.Angle_deg source_vAC_phase(start = 0) "Phase (deg) of drain AC small-
  signal source voltage";
SI.Voltage bulk_vAC_mag "Magnitude of AC small-signal bulk voltage";
SI.Voltage bulk_vAC_mag_dB "Magnitude (dB) of AC small-signal bulk voltage";
nonSI.Angle_deg bulk_vAC_phase(start = 0) "Phase (deg) of AC small-signal
  bulk voltage";
SI.Voltage gate_vAC_mag "Magnitude of AC small-signal gate voltage";
SI.Voltage gate_vAC_mag_dB "Magnitude (dB) of AC small-signal gate voltage";
nonSI.Angle_deg gate_vAC_phase(start = 0) "Phase (deg) of AC small-signal
  gate voltage";
SI.Current drain_iAC_mag "Magnitude of AC small-signal drain current";
SI.Current drain_iAC_mag_dB "Magnitude (dB) of AC small-signal drain
  current";
nonSI.Angle_deg drain_iAC_phase(start = 0) "Phase (deg) of AC small-signal
  drain current";
SI.Current source_iAC_mag "Magnitude of AC small-signal source current";
SI.Current source_iAC_mag_dB "Magnitude (dB) of AC small-signal source
  current";
nonSI.Angle_deg source_iAC_phase(start = 0) "Phase (deg) of drain AC small-
signal source current";
SI.Current bulk_iAC_mag "Magnitude of AC small-signal bulk current";
SI.Current bulk_iAC_mag_dB "Magnitude (dB) of AC small-signal bulk current";
nonSI.Angle_deg bulk_iAC_phase(start = 0) "Phase (deg) of AC small-signal
  bulk current";
SI.Current gate_iAC_mag "Magnitude of AC small-signal gate current";
SI.Current gate_iAC_mag_dB "Magnitude (dB) of AC small-signal gate current";
nonSI.Angle_deg gate_iAC_phase(start = 0) "Phase (deg) of AC small-signal
  gate current";
parameter SI.Length L = 1e-07 "Gate length";
parameter SI.Length W = 4e-06 "Gate width";
inner SI.Voltage vsdDCSgn "Source-pin to drain-pin voltage";
inner SI.Voltage vthDC "Threshold voltage";
inner SI.Voltage vsdTranSgn "Source-pin to drain-pin voltage";
inner SI.Voltage vthTran "Threshold voltage";
inner SI.Voltage gate_vAC_Re;
inner SI.Voltage gate_vAC_Im;
inner SI.Voltage bulk_vAC_Re;
inner SI.Voltage bulk_vAC_Im;

PMOS.Isd Isd1(VTO = VTO, GAMMA = GAMMA, PHI = PHI, KP = KP, W = W, L = L,
  LD = LD, LAMBDA = LAMBDA);
PMOS.Idiode Dsb(IS = IS);
PMOS.Cdiode Csb(CJ = CJ, CJSW = CJSW, MJ = MJ, MJSW = MJSW, FC = FC, PB =
  PB, P = PS, A = AS);
PMOS.Idiode Ddb(IS = IS);
PMOS.Cdiode Cdb(CJ = CJ, CJSW = CJSW, MJ = MJ, MJSW = MJSW, FC = FC, PB =
  PB, P = PD, A = AD);
PMOS.Cdg Cdg1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
  CGDO = CGDO, CGSO = CGSO, sourceGatePinsC = false);
PMOS.Cdg Csg1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
  CGDO = CGDO, CGSO = CGSO, sourceGatePinsC = true);
PMOS.Cbg Cbg1(PHI = PHI, LD = LD, W = W, L = L, TOX = TOX, EPSR = EPSR,
  CGBO = CGBO);
```

```
equation
  vthDC = VTO - GAMMA * (sqrt(abs(PHI - vsbDC)) - sqrt(PHI));
  vthTran = VTO - GAMMA * (sqrt(abs(PHI - vsbTran)) - sqrt(PHI));
  vsdDC = noEvent(abs(Isd1.vDC));
  vsdDCSgn = Isd1.vDC;
  vsdTran = noEvent(abs(Csg1.vTran - Cdg1.vTran));
  vsdTranSgn = Csg1.vTran - Cdg1.vTran;
  vsgDC = max({Csg1.vDC,Cdg1.vDC});
  vsgTran = max({Csg1.vTran,Cdg1.vTran});
  vsbDC = max({Csb.vDC,Cdb.vDC});
  vsbTran = max({Csb.vTran,Cdb.vTran});
  gate_vAC_Re = g.vAC_Re;
  gate_vAC_Im = g.vAC_Im;
  bulk_vAC_Re = b.vAC_Re;
  bulk_vAC_Im = b.vAC_Im;
  (drain_vAC_mag,drain_vAC_phase) = Rect2Polar({d.vAC_Re,d.vAC_Im});
  drain_vAC_mag_dB = Decibels(drain_vAC_mag);
  (drain_iAC_mag,drain_iAC_phase) = Rect2Polar({d.iAC_Re,d.iAC_Im});
  drain_iAC_mag_dB = Decibels(drain_iAC_mag);
  (source_vAC_mag,source_vAC_phase) = Rect2Polar({s.vAC_Re,s.vAC_Im});
  source_vAC_mag_dB = Decibels(source_vAC_mag);
  (source_iAC_mag,source_iAC_phase) = Rect2Polar({s.iAC_Re,s.iAC_Im});
  source_iAC_mag_dB = Decibels(source_iAC_mag);
  (bulk_vAC_mag,bulk_vAC_phase) = Rect2Polar({b.vAC_Re,b.vAC_Im});
  bulk_vAC_mag_dB = Decibels(bulk_vAC_mag);
  (bulk_iAC_mag,bulk_iAC_phase) = Rect2Polar({b.iAC_Re,b.iAC_Im});
  bulk_iAC_mag_dB = Decibels(bulk_iAC_mag);
  (gate_vAC_mag,gate_vAC_phase) = Rect2Polar({g.vAC_Re,g.vAC_Im});
  gate_vAC_mag_dB = Decibels(gate_vAC_mag);
  (gate_iAC_mag,gate_iAC_phase) = Rect2Polar({g.iAC_Re,g.iAC_Im});
  gate_iAC_mag_dB = Decibels(gate_iAC_mag);
  connect(d,Isd1.n);
  connect(Isd1.p,s);
  connect(g,Cdg1.n);
  connect(g,Csg1.n);
  connect(g,Cbg1.n);
  connect(b,Dsb.n);
  connect(b,Csb.n);
  connect(b,Ddb.n);
  connect(b,Cdb.n);
  connect(b,Cbg1.p);
  connect(Isd1.p,Csg1.p);
  connect(Isd1.n,Cdg1.p);
  connect(Isd1.p,Csb.p);
  connect(Isd1.n,Cdb.p);
  connect(Cdb.p,Ddb.p);
  connect(Dsb.p,Csb.p);
end PMOStransistor;


package PMOS "Contains the components used to model an PMOS transistor"
 extends Modelica.Icons.Library2;

model Idiode
 extends INTERFACE.OnePort;
 parameter SI.Current IS "Reverse saturation current at 300K";
 constant SI.Voltage thermalVolt = 300 / 11600 "Thermal voltage";
 constant SI.Conductance GMIN = 1e-12 "Conductance in parallel with the pn-
   junction";
 SI.Conductance gAC(start = 1) "AC small-signal conductance";

equation
 iDC = IS * (exp(vDC / thermalVolt) - 1) + vDC * GMIN;
 iTran = if noEvent(vTran > 0) then IS * (exp(vTran / thermalVolt) - 1) +
   vTran * GMIN else (IS / thermalVolt + GMIN) * vTran;
 gAC = if vDC > 0 then IS / thermalVolt * exp(vDC / thermalVolt) + GMIN else
   IS / thermalVolt + GMIN;
 {iAC_Re,iAC_Im} = gAC * {vAC_Re,vAC_Im};
```

```
end Idiode;


model Isd
  extends INTERFACE.OnePort;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vsgDC "Source to gate voltage";
  outer SI.Voltage vsbDC "Source to bulk voltage";
  outer SI.Voltage vsdDC "Source to drain voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vsgTran "Source to gate voltage";
  outer SI.Voltage vsbTran "Source to bulk voltage";
  outer SI.Voltage vsdTran "Source to drain voltage";
  outer SI.Voltage vsdTranSgn "Source-pin to drain-pin voltage";
  outer SI.Voltage gate_vAC_Re;
  outer SI.Voltage gate_vAC_Im;
  outer SI.Voltage bulk_vAC_Re;
  outer SI.Voltage bulk_vAC_Im;
  parameter SI.Voltage VTO "Zero-bias threshold voltage";
  parameter Real GAMMA "Body-effect parameter [V0.5]";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter Real KP "Transconductance parameter [A/V2]";
  parameter SI.Length W "Gate width";
  parameter SI.Length L "Gate length";
  parameter SI.Length LD "Lateral diffusion";
  parameter Real LAMBDA "Channel-length modulation [V-1]";
protected
  parameter Real beta = (KP * W) / (L - 2 * LD);
  Boolean pinPisSourceAC(start = true);
  SI.Current isdDC "Source to drain current";
  SI.Current isdTran "Source to drain current";
  Real ARG(start = 0);
  SI.Conductance gsdAC "dIsd/dVsd";
  SI.Conductance gmbsAC "dIsd/dVsb";
  SI.Conductance gmAC "dIsd/dVsg";
  SI.Voltage vsdAC_Re;
  SI.Voltage vsdAC_Im;
  SI.Voltage vsgAC_Re;
  SI.Voltage vsgAC_Im;
  SI.Voltage vsbAC_Re;
  SI.Voltage vsbAC_Im;
  SI.Current isdAC_Re;
  SI.Current isdAC_Im;

equation
  iDC = noEvent(if vDC > 0 then isdDC else -isdDC);
  isdDC = if vsgDC <= -vthDC then 0 else noEvent(if vsdDC < vsgDC + vthDC
    then beta * (vsgDC + vthDC - 0.5 * vsdDC) * vsdDC * (1 + LAMBDA * vsdDC)
  else 0.5 * beta * (vsgDC + vthDC) ^ 2 * (1 + LAMBDA * vsdDC));
  iTran = if noEvent(vsdTranSgn >= 0) then isdTran else -isdTran;
  isdTran = if vsgTran <= -vthTran then 0 else if noEvent(vsdTran < vsgTran +
    vthTran) then beta * (vsgTran + vthTran - 0.5 * vsdTran) * vsdTran * (1 +
    LAMBDA * vsdTran) else 0.5 * beta * (vsgTran + vthTran) ^ 2 * (1 + LAMBDA
    * vsdTran);
  {vsdAC_Re,vsdAC_Im} = if pinPisSourceAC then {vAC_Re,vAC_Im} else -
    {vAC_Re,vAC_Im};
  {vsgAC_Re,vsgAC_Im} = if pinPisSourceAC then -{gate_vAC_Re -
    p.vAC_Re,gate_vAC_Im - p.vAC_Im} else -{gate_vAC_Re -
    n.vAC_Re,gate_vAC_Im - n.vAC_Im};
  {vsbAC_Re,vsbAC_Im} = if pinPisSourceAC then -{bulk_vAC_Re -
    p.vAC_Re,bulk_vAC_Im - p.vAC_Im} else -{bulk_vAC_Re -
    n.vAC_Re,bulk_vAC_Im - n.vAC_Im};
  {isdAC_Re,isdAC_Im} = gsdAC * {vsdAC_Re,vsdAC_Im} + gmbsAC *
    {vsbAC_Re,vsbAC_Im} + gmAC * {vsgAC_Re,vsgAC_Im};
  {iAC_Re,iAC_Im} = if pinPisSourceAC then {isdAC_Re,isdAC_Im} else -
    {isdAC_Re,isdAC_Im};
  pinPisSourceAC = vDC >= 0;
  gsdAC = if vsgDC <= -vthDC then 0 else if vsdDC < vsgDC + vthDC then beta *
```

```
    (vsgDC + vthDC - vsdDC) * (1 + LAMBDA * vsdDC) + beta * (vsgDC + vthDC -
    0.5 * vsdDC) * LAMBDA * vsdDC else 0.5 * beta * (vsgDC + vthDC) ^ 2 *
    LAMBDA;
  gmbsAC = gmAC * ARG;
  ARG = if vsbDC > 0 then (0.5 * GAMMA) / (sqrt(PHI) - (0.5 * vsbDC) /
    sqrt(PHI)) else (0.5 * GAMMA) / sqrt(PHI - vsbDC);
  gmAC = if vsgDC <= -vthDC then 0 else if vsdDC < vsgDC + vthDC then beta *
    vsdDC * (1 + LAMBDA * vsdDC) else beta * (vsgDC + vthDC) * (1 + LAMBDA *
    vsdDC);
end Isd;


partial model Capacitor
  extends INTERFACE.OnePort;
  constant Real pi = 3.14159265358979;
  outer SI.Frequency freq "Frequency of the source";
  SI.Capacitance Cvar(start = 1e-12) "Transient analysis capacitance";
  SI.Capacitance CvarAC(start = 1e-12) "AC small-signal capacitance";

equation
  iDC = 0;
  {vAC_Re,vAC_Im} * 2 * pi * freq * CvarAC = {iAC_Im,-iAC_Re};
end Capacitor;


model Cdiode
  extends Capacitor;
  parameter Real CJ "Capacitance at zero-bias voltage per squere meter of
    area [F/m2]";
  parameter Real CJSW "Capacitance at zero-bias voltage per meter of
    perimeter [F/m]";
  parameter Real MJ "Bulk junction capacitnce grading coefficient";
  parameter Real MJSW "Perimeter capacitance grading coefficient";
  parameter Real FC "Substrate-junction forward-bias coefficient";
  parameter SI.Voltage PB "Junction potencial";
  parameter SI.Length P "Junction perimeter";
  parameter SI.Area A "Junction area";
protected
  parameter Real F2 = (1 - FC) ^ (1 + MJ);
  parameter Real F3 = 1 - FC * (1 + MJ);
  parameter Real FCtimesPB = FC * PB;
  parameter Real CJtimesA = CJ * A;
  parameter Real CJSWtimesP = CJSW * P;
  parameter Real coef1 = F3 * (CJtimesA / F2 + (CJSW * P) / F2);
  parameter Real coef2 = ((CJtimesA * MJ) / F2 + (CJSW * P * MJSW) / F2) / PB;
  parameter Real invPB = 1 / PB;

equation
  Cvar = if noEvent(vTran < FCtimesPB) then CJtimesA / (1 - invPB * vTran) ^
    MJ + CJSWtimesP / (1 - invPB * vTran) ^ MJSW else coef1 + coef2 * vTran;
  Cvar * der(vTran) = iTran;
  CvarAC = if vDC < FCtimesPB then CJtimesA / (1 - invPB * vDC) ^ MJ +
    CJSWtimesP / (1 - invPB * vDC) ^ MJSW else coef1 + coef2 * vDC;
end Cdiode;


model Cbg
  extends Capacitor;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vsdDC "Source to drain voltage";
  outer SI.Voltage vsgDC "Source to gate voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vsdTran "Source to drain voltage";
  outer SI.Voltage vsgTran "Source to gate voltage";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter SI.Length LD "Lateral diffusion";
  parameter SI.Length W "Gate width";
  parameter SI.Length L "Gate length";
```

```
  parameter SI.Length TOX "Gate oxide thickness";
  parameter Real EPSR "Dielectric constant of the oxide";
  parameter Real CGBO "Gate-bulk overlap capacitance per meter [F/m]";
protected
  parameter SI.Length LEFF = L + 2 * LD "Effective length";
  parameter SI.Capacitance COX = (EPS0 * EPSR * W * LEFF) / TOX "Gate oxide
  capacitance";
  parameter Real CGBOtimesLEFF = CGBO * LEFF;
  parameter Real COXtimesinvPHI = COX / PHI;
  constant Real EPS0 = 8.85e-12 "Permittivity of free space [F/m]";

equation
  Cvar - CGBOtimesLEFF = if noEvent(vsgTran < -vthTran - PHI) then COX else
    if noEvent(vsgTran < -vthTran) then COXtimesinvPHI * (-vthTran - vsgTran)
    else 0;
  Cvar * der(vTran) = iTran;
  CvarAC = if vsgDC < -vthDC - PHI then COX + CGBOtimesLEFF else if vsgDC < -
    vthDC then COXtimesinvPHI * (-vthDC - vsgDC) + CGBOtimesLEFF else
    CGBOtimesLEFF;
end Cbg;


model Cdg
  extends Capacitor;
  outer SI.Voltage vthDC "Threshold voltage";
  outer SI.Voltage vsdDC "Source to drain voltage";
  outer SI.Voltage vsdDCSgn "Source-pin to drain-pin voltage";
  outer SI.Voltage vsgDC "Source to gain voltage";
  outer SI.Voltage vthTran "Threshold voltage";
  outer SI.Voltage vsdTran "Source to drain voltage";
  outer SI.Voltage vsdTranSgn "Source-pin to drain-pin voltage";
  outer SI.Voltage vsgTran "Source to gate voltage";
  parameter SI.Voltage PHI "Surface inversion potencial";
  parameter SI.Length LD "Lateral diffusion";
  parameter SI.Length W "Gate width";
  parameter SI.Length L "Gate length";
  parameter SI.Length TOX "Gate oxide thickness";
  parameter Real EPSR "Dielectric constant of the oxide";
  parameter Real CGDO "Gate-drain overlap capacitance per meter [F/m]";
  parameter Real CGSO "Gate-source overlap capacitance per meter [F/m]";
  parameter Boolean sourceGatePinsC;
protected
  parameter SI.Length LEFF = L + 2 * LD "Effective length";
  parameter SI.Capacitance COX = (EPS0 * EPSR * W * LEFF) / TOX "Gate oxide
    capacitance";
  constant Real EPS0 = 8.85e-12 "Permittivity of free space [F/m]";
  parameter Real CGSOtimesW = CGSO * W;
  parameter Real CGDOtimesW = CGDO * W;
  parameter Real twoThirdsCOX = 2 / 3 * COX;
  parameter Real threeFourthsCOX = 3 / 4 * COX;
  parameter Real halfCOX = 0.5 * COX;
  parameter SI.Voltage vsdTranEPS = 0.0001;
  parameter SI.Voltage vsdDCEPS = 0.0001;
  Real Csg;
  Real Cdg;
  Real CsgAC;
  Real CdgAC;
  Real CpinsSG;
  Real CpinsDG;
  Real CpinsSGAC;
  Real CpinsDGAC;

equation
  Cvar * der(vTran) = iTran;
  Cvar = if sourceGatePinsC then CpinsSG else CpinsDG;
  CpinsSG = if noEvent(vsdTranSgn < -vsdTranEPS) then Cdg else if
    noEvent(vsdTranSgn < vsdTranEPS) then (0.5 * (Csg - Cdg) * vsdTranSgn) /
    vsdTranEPS + 0.5 * (Csg + Cdg) else Csg;
```

```
  CpinsDG = if noEvent(vsdTranSgn < -vsdTranEPS) then Csg else if
    noEvent(vsdTranSgn < vsdTranEPS) then (0.5 * (Cdg - Csg) * vsdTranSgn) /
    vsdTranEPS + 0.5 * (Csg + Cdg) else Cdg;
  Csg - CGSOtimesW = if noEvent(vsgTran <= -vthTran - PHI) then 0 else if
    noEvent(vsgTran <= -vthTran) then twoThirdsCOX * ((vthTran + vsgTran) /
    PHI + 1) else if noEvent(vsgTran <= vthTran + vsdTran) then twoThirdsCOX
    else if noEvent(vsdTran <= 2 * vsdTranEPS) then halfCOX else twoThirdsCOX
    * (1 - ((vsgTran - vsdTran + vthTran) / (2 * (vsgTran + vthTran) -
    vsdTran)) ^ 2);
  Cdg - CGDOtimesW = if noEvent(vsgTran <= -vthTran + vsdTran) then 0 else if
    noEvent(vsdTran <= 2 * vsdTranEPS) then threeFourthsCOX else COX * (1 -
    ((vsgTran + vthTran) / (2 * (vsgTran + vthTran) - vsdTran)) ^ 2);
  CvarAC = if sourceGatePinsC then CpinsSGAC else CpinsDGAC;
  CpinsSGAC = if noEvent(vsdDCSgn < -vsdDCEPS) then CdgAC else if
    noEvent(vsdDCSgn < vsdDCEPS) then (0.5 * (CsgAC - CdgAC) * vsdDCSgn) /
    vsdDCEPS + 0.5 * (CsgAC + CdgAC) else CsgAC;
  CpinsDGAC = if noEvent(vsdDCSgn < -vsdDCEPS) then CsgAC else if
    noEvent(vsdDCSgn < vsdDCEPS) then (0.5 * (CdgAC - CsgAC) * vsdDCSgn) /
    vsdDCEPS + 0.5 * (CsgAC + CdgAC) else CdgAC;
  CsgAC = if vsgDC <= -vthDC - PHI then CGSOtimesW else if vsgDC <= -vthDC
    then twoThirdsCOX * ((vthDC + vsgDC) / PHI + 1) + CGSOtimesW else if
    vsgDC <= -vthDC + vsdDC then twoThirdsCOX + CGSOtimesW else if vsdDC <= 2
    * vsdDCEPS then halfCOX + CGSOtimesW else twoThirdsCOX * (1 - ((vsgDC -
    vsdDC + vthDC) / (2 * (vsgDC + vthDC) - vsdDC)) ^ 2) + CGSOtimesW;
  CdgAC = if vsgDC <= -vthDC + vsdDC then CGDOtimesW else if vsdDC <= 2 *
    vsdDCEPS then threeFourthsCOX + CGDOtimesW else COX * (1 - ((vsgDC +
    vthDC) / (2 * (vsgDC + vthDC) - vsdDC)) ^ 2) + CGDOtimesW;
end Cdg;
end PMOS;

end Components;
```

Constants sub-package

```
package Constants "Contains constants and fabrication parameters for
  different components"
  extends Modelica.Icons.Library2;
  import SI = Modelica.SIunits;

package ResistorConstants "Contains constants and parameters for resistor"
  final constant SI.Resistance Rs = 50 "Sheet resistance of the resistor
  material";
end ResistorConstants;

package CapacitorConstants "Contains constants and parameters for capacitor"
  final constant Real pi = 2 * Modelica.Math.asin(1.0);
  final constant Real Er = 3.4 "relative permittivity of capacitor
    dielectric";
  final constant SI.Distance t = 5e-08 "Separation of capacitor metal plates";
end CapacitorConstants;

package InductorConstants "Contains constants and parameters for inductor"
  final constant Real pi = 2 * Modelica.Math.asin(1.0);
end InductorConstants;

package NMOStransistorConstants "Contains constants and parameters for NMOS
  transistor"
  constant SI.Area AD = 0 "drain difussion area";
  constant SI.Area AS = 0 "Source difussion area";
  constant SI.Permittivity CGBO = 3e-10 "Gate-bulk overlap capacitance per
    meter [F/m]";
  constant SI.Permittivity CGDO = 3e-10 "Gate-drain overlap capacitance per
    meter [F/m]";
  constant SI.Permittivity CGSO = 3e-10 "Gate-source overlap capacitance per
```

```
    meter [F/m]";
  constant Real CJ = 0.0001 "Capacitance at zero-bias voltage per square
    meter of area [F/m2]";
  constant SI.Permittivity CJSW = 5e-10 "Capacitance at zero-bias voltage per
    meter of perimeter [F/m]";
  constant Real FC = 0.5 "Substrate-junction forward-bias coefficient";
  constant Real GAMMA = 0.4675 "Body-effect parameter [V0.5]";
  constant SI.Current IS = 1e-14 "Reverse saturation current at 300K";
  constant Real KP = 5e-05 "Transconductance parameter [A/V2]";
  constant Real LAMBDA = 0.033 "Channel-length modulation [V-1]";
  constant SI.Length LD = 0 "Lateral diffusion";
  constant Real MJ = 0.75 "Bulk junction capacitnce grading coefficient";
  constant Real MJSW = 0.33 "Perimeter capacitance grading coefficient";
  constant SI.Length PD = 0.0004 "drain difussion perimeter";
  constant SI.Length PS = 0.0004 "source difussion perimeter";
  constant SI.Voltage PB = 0.95 "Junction potencial";
  constant SI.Voltage PHI = 0.6 "Surface inversion potencial";
  constant SI.Length TOX = 4.2e-08 "Gate oxide thickness";
  constant SI.Voltage VTO = 0.43 "Zero-bias threshold voltage";
  constant Real EPSR = 3.9 "Dielectric constant of the oxide";
end NMOStransistorConstants;

package PMOStransistorConstants "Contains constants and parameters for NMOS
  transistor"
  constant SI.Area AD = 0 "drain difussion area";
  constant SI.Area AS = 0 "Source difussion area";
  constant SI.Permittivity CGBO = 3e-10 "Gate-bulk overlap capacitance per
    meter [F/m]";
  constant SI.Permittivity CGDO = 3e-10 "Gate-drain overlap capacitance per
    meter [F/m]";
  constant SI.Permittivity CGSO = 3e-10 "Gate-source overlap capacitance per
    meter [F/m]";
  constant Real CJ = 0.0001 "Capacitance at zero-bias voltage per square
    meter of area [F/m2]";
  constant SI.Permittivity CJSW = 5e-10 "Capacitance at zero-bias voltage per
    meter of perimeter [F/m]";
  constant Real FC = 0.5 "Substrate-junction forward-bias coefficient";
  constant Real GAMMA = 0.4675 "Body-effect parameter [V0.5]";
  constant SI.Current IS = 1e-14 "Reverse saturation current at 300K";
  constant Real KP = 5e-05 "Transconductance parameter [A/V2]";
  constant Real LAMBDA = 0.033 "Channel-length modulation [V-1]";
  constant SI.Length LD = 0 "Lateral diffusion";
  constant Real MJ = 0.75 "Bulk junction capacitnce grading coefficient";
  constant Real MJSW = 0.33 "Perimeter capacitance grading coefficient";
  constant SI.Length PD = 0.0004 "drain difussion perimeter";
  constant SI.Length PS = 0.0004 "source difussion perimeter";
  constant SI.Voltage PB = 0.95 "Junction potencial";
  constant SI.Voltage PHI = 0.6 "Surface inversion potencial";
  constant SI.Length TOX = 4.2e-08 "Gate oxide thickness";
  constant SI.Voltage VTO = -0.43 "Zero-bias threshold voltage";
  constant Real EPSR = 3.9 "Dielectric constant of the oxide";
end PMOStransistorConstants;

end Constants;
```

Examples sub-package

```
package Examples "Contains different examples of common circuits"
  extends Modelica.Icons.Library2;

model CMOSInverter
  inner Modelica.SIunits.Frequency freq = 10;

  Modelica.Electrical.IC.Components.Ground ground4;
```

```
  Modelica.Electrical.IC.Components.Ground ground3;
  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Components.Ground ground2;
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1(L = 1e-07,
    W = 4e-06);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 1,
    phase = 0, DCVoltage = 0.6, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 0, V1 = 1.2);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0, riseTime = 0, fallTime = 0,
    stepWidth = 0, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Components.Capacitor capacitor1(W = 4e-06, L = 4e-
    06);
  Modelica.Electrical.IC.Components.PMOStransistor pmostransistor1(L = 1e-07,
    W = 4e-06);

equation
  connect(capacitor1.n,ground2.p);
  connect(capacitor1.p,nmostransistor1.d);
  connect(pmostransistor1.d,nmostransistor1.d);
  connect(voltagesource2.n,ground4.p);
  connect(voltagesource2.p,nmostransistor1.g);
  connect(pmostransistor1.b,pmostransistor1.s);
  connect(pmostransistor1.g,nmostransistor1.g);
  connect(voltagesource1.p,pmostransistor1.s);
  connect(nmostransistor1.s,ground1.p);
  connect(voltagesource1.n,ground3.p);
  connect(nmostransistor1.b,nmostransistor1.s);
end CMOSInverter;


model CommonDrainStage
  inner Modelica.SIunits.Frequency freq = 100;

  Modelica.Electrical.IC.Components.Ground ground2;
  Modelica.Electrical.IC.Components.Ground ground3;
  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0, riseTime = 0, fallTime = 0,
    stepWidth = 1, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Components.Ground ground4;
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1(L = 1e-07,
    W = 4e-06);
  Modelica.Electrical.IC.Components.Capacitor capacitor1(W = 4e-06, L = 4e-
    06);
  Modelica.Electrical.IC.Sources.CurrentSource currentsource1(Amplitude = 0,
    phase = 0, DCcurrent = 1e-05, delay = 0, riseTime = 1, fallTime = 1,
    stepWidth = 10, I0 = 1e-05, I1 = 1e-05);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 1,
    phase = 0, DCVoltage = 1.0, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 1, V1 = 1.2);

equation
  connect(nmostransistor1.b,ground3.p);
  connect(voltagesource2.n,ground1.p);
  connect(nmostransistor1.g,voltagesource1.p);
  connect(voltagesource1.n,ground2.p);
  connect(voltagesource2.p,nmostransistor1.d);
  connect(nmostransistor1.s,currentsource1.p);
  connect(currentsource1.n,ground3.p);
  connect(capacitor1.p,nmostransistor1.s);
  connect(capacitor1.n,ground4.p);
end CommonDrainStage;


model CommonSourceStage
 inner Modelica.SIunits.Frequency freq = 100;
```

```
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1(L = 1e-07,
    W = 2e-05);
  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Components.Capacitor capacitor1(W = 0.00012, L =
    0.00012);
  Modelica.Electrical.IC.Components.Resistor resistor1(W = 0.001, L = 0.03);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 1,
    phase = 0, DCVoltage = 0.7, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 0, V1 = 1);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0, riseTime = 0, fallTime = 0,
    stepWidth = 1, V0 = 1.2, V1 = 1.2);

equation
  connect(voltagesource2.p,nmostransistor1.g);
  connect(voltagesource2.n,ground1.p);
  connect(voltagesource1.n,ground1.p);
  connect(voltagesource1.p,resistor1.p);
  connect(nmostransistor1.b,nmostransistor1.s);
  connect(nmostransistor1.s,ground1.p);
  connect(ground1.p,capacitor1.n);
  connect(nmostransistor1.d,capacitor1.p);
  connect(resistor1.n,nmostransistor1.d);
end CommonSourceStage;


model CurrentMirror
  inner Modelica.SIunits.Frequency freq = 100;

  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Components.Ground ground2;
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor2(L = 1e-07,
    W = 4e-06);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 1, riseTime = 10, fallTime = 10,
    stepWidth = 100, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Sources.CurrentSource currentsource1(Amplitude = 1,
    phase = 0, DCcurrent = 1e-05, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, I0 = 0, I1 = 1);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 0,
    phase = 0, DCVoltage = 0.4, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 0.4, V1 = 0.4);
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1(L = 1e-07,
    W = 4e-06);

equation
  connect(nmostransistor1.b,nmostransistor1.s);
  connect(nmostransistor1.s,ground2.p);
  connect(nmostransistor2.s,ground1.p);
  connect(currentsource1.p,voltagesource1.p);
  connect(voltagesource2.p,voltagesource1.p);
  connect(nmostransistor1.g,nmostransistor2.g);
  connect(voltagesource2.n,nmostransistor1.d);
  connect(currentsource1.n,nmostransistor2.d);
  connect(nmostransistor2.g,nmostransistor2.d);
  connect(nmostransistor2.b,nmostransistor2.s);
  connect(voltagesource1.n,ground2.p);
end CurrentMirror;


model TransmissionGate
  inner Modelica.SIunits.Frequency freq = 10;

  Modelica.Electrical.IC.Components.Ground ground4;
  Modelica.Electrical.IC.Components.Ground ground3;
  Modelica.Electrical.IC.Components.Ground ground2;
  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Components.PMOStransistor pmostransistor1;
```

```
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1;
  Modelica.Electrical.IC.Components.Capacitor capacitor1;
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 0,
    phase = 0, DCVoltage = 0.6, delay = 0.001, riseTime = 500, fallTime = 1,
    stepWidth = 1, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource4(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource3(Amplitude = 0,
    phase = 0, DCVoltage = 0, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 0, V1 = 1.2);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 1.2, V1 = 0);

equation
  connect(voltagesource2.p,nmostransistor1.g);
  connect(voltagesource2.n,ground3.p);
  connect(voltagesource3.n,ground4.p);
  connect(voltagesource3.p,pmostransistor1.g);
  connect(nmostransistor1.s,capacitor1.p);
  connect(ground1.p,nmostransistor1.b);
  connect(voltagesource4.n,ground1.p);
  connect(voltagesource4.p,pmostransistor1.b);
  connect(nmostransistor1.d,voltagesource1.p);
  connect(voltagesource1.p,pmostransistor1.s);
  connect(voltagesource1.n,ground2.p);
  connect(capacitor1.n,ground3.p);
  connect(pmostransistor1.d,capacitor1.p);
end TransmissionGate;


model TwoStage
  inner Modelica.SIunits.Frequency freq = 10;

  Modelica.Electrical.IC.Components.Ground ground3;
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor1;
  Modelica.Electrical.IC.Components.Ground ground2;
  Modelica.Electrical.IC.Components.Ground ground4;
  Modelica.Electrical.IC.Components.Ground ground1;
  Modelica.Electrical.IC.Components.NMOStransistor nmostransistor2(L = 1e-07,
    W = 4e-06);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource2(Amplitude = 0,
    phase = 0, DCVoltage = 1.2, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 1.2, V1 = 1.2);
  Modelica.Electrical.IC.Sources.VoltageSource voltagesource1(Amplitude = 1,
    phase = 0, DCVoltage = 0.9, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, V0 = 0, V1 = 1);
  Modelica.Electrical.IC.Components.Resistor resistor1(W = 0.001, L = 0.06);
  Modelica.Electrical.IC.Sources.CurrentSource currentsource1(Amplitude = 0,
    phase = 0, DCcurrent = 1e-05, delay = 0.1, riseTime = 50, fallTime = 50,
    stepWidth = 200, I0 = 1e-05, I1 = 1e-05);
  Modelica.Electrical.IC.Components.Ground ground5;
  Modelica.Electrical.IC.Components.Capacitor capacitor1(W = 4e-05, L = 4e-
  05);

equation
  connect(nmostransistor1.s,capacitor1.p);
  connect(capacitor1.n,ground4.p);
  connect(ground5.p,nmostransistor2.s);
  connect(nmostransistor1.b,ground3.p);
  connect(voltagesource2.p,nmostransistor1.d);
  connect(nmostransistor2.d,nmostransistor1.g);
  connect(currentsource1.p,nmostransistor1.s);
  connect(currentsource1.n,ground3.p);
  connect(nmostransistor2.b,nmostransistor2.s);
  connect(resistor1.p,voltagesource2.p);
  connect(resistor1.n,nmostransistor2.d);
```

```
  connect(voltagesource1.p,nmostransistor2.g);
  connect(voltagesource1.n,ground2.p);
  connect(voltagesource2.n,ground1.p);
end TwoStage;

end Examples;
```

Functions sub-package

```
package Functions "Contains functions for units and coordinates translation"
  extends Modelica.Icons.Library2;
  import SI = Modelica.SIunits;
  import nonSI = Modelica.SIunits.Conversions.NonSIunits;
  import Modelica.Math;

function Rect2Polar
  input Real rect[2];
  output Real module;
  output nonSI.Angle_deg angle;
protected
  SI.Angle angFirst;
  constant Real pi = 3.14159265358979;
algorithm
  module:=sqrt(rect * rect);
  angFirst:=if not module > 0 then 0 else Math.asin(abs(rect[2]) / module);
  angle:=if rect[1] < 0 then pi - angFirst else angFirst;
  angle:=if rect[2] < 0 then -angle else angle;
  angle:=(180 * angle) / pi;
end Rect2Polar;


function Decibels
  input Real x;
  output Real x_dB;
algorithm
  x_dB:= 20 * Math.log10(x);
end Decibels;


function Rad2Deg
  input SI.Angle angle_rad;
  output nonSI.Angle_deg angle_deg;
protected
  constant Real pi = 3.14159265358979;
algorithm
  angle_deg:=(180 * angle_rad) / pi;
end Rad2Deg;


function Deg2Rad
  input nonSI.Angle_deg angle_deg;
  output SI.Angle angle_rad;
protected
  constant Real pi = 3.14159265358979;
algorithm
  angle_rad:=(pi * angle_deg) / 180;
end Deg2Rad;

end Functions;
```

Interfaces sub-package

```
package Interfaces "Needed connectors for IC design"
  extends Modelica.Icons.Library2;
  import SI = Modelica.SIunits;

connector PositivePin
  SI.Voltage vDC "DC potential at the pin";
  SI.Voltage vTran "Transient/Small-signal potential at the pin";
  SI.Voltage vAC_Re "Small-signal potential at the pin. Real part";
  SI.Voltage vAC_Im "Small-signal potential at the pin. Imaginary part";
  flow SI.Current iDC "DC current flowing into the pin";
  flow SI.Current iTran "Transient current flowing into the pin";
  flow SI.Current iAC_Re "Small-signal current flowing into the pin. Real
    part";
  flow SI.Current iAC_Im "Small-signal current flowing into the pin.
    Imaginary part";
end PositivePin;

connector NegativePin
  SI.Voltage vDC "DC potential at the pin";
  SI.Voltage vTran "Transient/Small-signal potential at the pin";
  SI.Voltage vAC_Re "Small-signal potential at the pin. Real part";
  SI.Voltage vAC_Im "Small-signal potential at the pin. Imaginary part";
  flow SI.Current iDC "DC current flowing into the pin";
  flow SI.Current iTran "Transient current flowing into the pin";
  flow SI.Current iAC_Re "Small-signal current flowing into the pin. Real
    part";
  flow SI.Current iAC_Im "Small-signal current flowing into the pin.
    Imaginary part";
end NegativePin;


partial model TwoPin
  Pin p "(+) node";
  Pin n "(-) node";
end TwoPin;


partial model MOSFET
  Pin d "Drain";
  Pin s "Source";
  Pin g "Gate";
  Pin b "Bulk";
end MOSFET;


partial model OnePort
  extends TwoPin;
  SI.Voltage vDC "DC voltage between pines";
  SI.Voltage vTran "Transient voltage between pines";
  SI.Voltage vAC_Re "Real part of AC small-signal voltage between pines";
  SI.Voltage vAC_Im "Imaginary part of AC small-signal voltage between pines";
  SI.Current iDC "DC current";
  SI.Current iTran "Transient/Small-signal current";
  SI.Current iAC_Re "Small-signal current. Real part";
  SI.Current iAC_Im "Small-signal current. Imaginary part";

equation
  {iDC,iTran,iAC_Re,iAC_Im} = {p.iDC,p.iTran,p.iAC_Re,p.iAC_Im};
  {iDC,iTran,iAC_Re,iAC_Im} = -{n.iDC,n.iTran,n.iAC_Re,n.iAC_Im};
  {vDC,vTran,vAC_Re,vAC_Im} = {p.vDC,p.vTran,p.vAC_Re,p.vAC_Im} -
    {n.vDC,n.vTran,n.vAC_Re,n.vAC_Im};
end OnePort;

end Interfaces;
```

Sources sub-package

```
package Sources "Contains voltage and current sources"
  extends Modelica.Icons.Library2;
  import SI = Modelica.SIunits;
  import nonSI = Modelica.SIunits.Conversions.NonSIunits;
  import Modelica.Electrical.IC.Functions.*;
  import Modelica.Math;
  import Interfaces = Modelica.Electrical.IC.Interfaces;

model VoltageSource "voltage source"
  extends Interfaces.OnePort;
  parameter SI.Voltage Amplitude(start = 1) "Amplitude AC signal";
  parameter nonSI.Angle_deg phase = 0 "Phase of AC signal";
  parameter SI.Voltage DCVoltage = 1.2 "DC offset";
  parameter Real delay = 0.1 "Time (in microsecs) when transient step rises";
  parameter Real riseTime = 50 "Time (in nanosecs) which takes signal going
    from V0 to V1";
  parameter Real fallTime = 50 "Time (in nanosecs) which takes signal going
    from V1 to V0";
  parameter Real stepWidth = 200 "Time (in nanosecs) since signal reachs V1
    until it start to fall towards V0";
  parameter SI.Voltage V0 = 0 "Low level voltage of the step";
  parameter SI.Voltage V1 = 1 "High level voltage of the step";
  SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage";
  SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
    across the component";
  SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
  SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
    current";
  SI.Conversions.NonSIunits.Angle_deg iAC_phase(start = 0) "Phase (deg) of AC
    small-signal current";
  SI.Voltage pinP_vAC_mag(start = 0);
  nonSI.Angle_deg pinP_vAC_phase(start = 0);
  SI.Voltage pinP_vAC_mag_dB(start = 0);
  SI.Voltage pinN_vAC_mag(start = 0);
  nonSI.Angle_deg pinN_vAC_phase(start = 0);
  SI.Voltage pinN_vAC_mag_dB(start = 0);

equation
  vDC = DCVoltage;
  vAC_mag = Amplitude;
  vAC_mag_dB = Decibels(vAC_mag);
  {vAC_Re,vAC_Im} = {Amplitude * cos(Deg2Rad(phase)),Amplitude *
    sin(Deg2Rad(phase))};
  (iAC_mag,iAC_phase) = Rect2Polar({iAC_Re,iAC_Im});
  iAC_mag_dB = Decibels(iAC_mag);
  (pinP_vAC_mag,pinP_vAC_phase) = Rect2Polar({p.vAC_Re,p.vAC_Im});
  pinP_vAC_mag_dB = Decibels(pinP_vAC_mag);
  (pinN_vAC_mag,pinN_vAC_phase) = Rect2Polar({n.vAC_Re,n.vAC_Im});
  pinN_vAC_mag_dB = Decibels(pinN_vAC_mag);
  assert(riseTime + fallTime < stepWidth, "Rise time + fall time is bigger
    than step width in voltage source parameters");
  vTran = if time < delay * 1e-06 then V0 else if time < delay * 1e-06 +
    riseTime * 1e-09 then V0 + (V1 - V0) / (riseTime * 1e-09) * (time - delay
    * 1e-06) else if time < delay * 1e-06 + riseTime * 1e-09 + stepWidth *
    1e-09 then V1 else if time < delay * 1e-06 + riseTime * 1e-09 + stepWidth
    * 1e-09 + fallTime * 1e-09 then V1 - (V1 - V0) / (fallTime * 1e-09) *
    (time - delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09) else V0;
end VoltageSource;


model CurrentSource "Current source"
  extends Interfaces.OnePort;
  parameter SI.Current Amplitude(start = 1) "Amplitude AC signal";
  parameter nonSI.Angle_deg phase = 0 "Phase of AC signal";
  parameter SI.Current DCcurrent = 1e-05 "DC offset";
```

```
  parameter Real delay = 0.1 "Time (in microsecs) when transient step rises";
  parameter Real riseTime = 50 "Time (in nanosecs) which takes signal going
    from V0 to V1";
  parameter Real fallTime = 50 "Time (in nanosecs) which takes signal going
    from V1 to V0";
  parameter Real stepWidth = 200 "Time (in nanosecs) since signal reachs V1
    until it start to fall towards V0";
  parameter SI.Current I0 = 0 "Low level current of the step";
  parameter SI.Current I1 = 1 "High level current of the step";
  SI.Voltage vAC_mag(start = 0) "Magnitude of AC small-signal voltage";
  SI.Voltage vAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal voltage
    across the component";
  SI.Conversions.NonSIunits.Angle_deg vAC_phase(start = 0) "Phase (deg) of AC
    small-signal voltage";
  SI.Current iAC_mag(start = 0) "Magnitude of AC small-signal current";
  SI.Current iAC_mag_dB(start = 0) "Magnitude (dB) of AC small-signal
    current";
  SI.Voltage pinP_iAC_mag(start = 0);
  nonSI.Angle_deg pinP_iAC_phase(start = 0);
  SI.Voltage pinP_iAC_mag_dB(start = 0);
  SI.Voltage pinN_iAC_mag(start = 0);
  nonSI.Angle_deg pinN_iAC_phase(start = 0);
  SI.Voltage pinN_iAC_mag_dB(start = 0);

equation
  iDC = DCcurrent;
  iAC_mag = Amplitude;
  iAC_mag_dB = Decibels(vAC_mag);
  {iAC_Re,iAC_Im} = {Amplitude * cos(Deg2Rad(phase)),Amplitude *
    sin(Deg2Rad(phase))};
  (vAC_mag,vAC_phase) = Rect2Polar({vAC_Re,vAC_Im});
  vAC_mag_dB = Decibels(vAC_mag);
  (pinP_iAC_mag,pinP_iAC_phase) = Rect2Polar({p.iAC_Re,p.iAC_Im});
  pinP_iAC_mag_dB = Decibels(pinP_iAC_mag);
  (pinN_iAC_mag,pinN_iAC_phase) = Rect2Polar({n.iAC_Re,n.iAC_Im});
  pinN_iAC_mag_dB = Decibels(pinN_iAC_mag);
  assert(riseTime + fallTime < stepWidth, "Rise time + Fall time is bigger
    than step width in current source parameters");
  iTran = if time < delay * 1e-06 then I0 else if time > delay * 1e-06 and
    time < delay * 1e-06 + riseTime * 1e-09 then (I1 - I0) / riseTime * 1e-09
    * (time - delay * 1e-06) else if time > delay * 1e-06 + riseTime * 1e-09
    and time < delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 then I1
    else if time > delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 and
    time < delay * 1e-06 + riseTime * 1e-09 + stepWidth * 1e-09 + fallTime *
    1e-09 then I1 - (I1 - I0) / fallTime * 1e-09 * (time - delay * 1e-06 +
    riseTime * 1e-09 + stepWidth * 1e-09) else I0;
end CurrentSource;

end Sources;


end IC;
```

Appendix B – Python Code

testModel_run.py

This is the principal module which should be executed.

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%               File name: testModel_run.py
#%               last modified 2011-07-06 david
#%               2011 - David Malo Cid
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#
# This script calls the make_OMscript.py and sim.py
#
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#
# load modules
# ----------------------------------------------------------------------------
import sys,os, random, time
global sweepVariable, jj, max_iter
max_iter = 7

now1= time.time()
#removing the simualtion files so that a fresh simualtion can start and in
#case of failure. Also changing the working directory
# ----------------------------------------------------------------------------
os.chdir(r'D:\David\project\ParametricSweep')
os.system('del test_res.plt')
os.system('del ParametricSweep.plt')
os.system('del test_init.txt')
fileObject26=open('ParametricSweep.plt','a+',1)
fileObject26.write("#Ptolemy Plot file, generated by OpenModelica" + "\n")
fileObject26.write("#NumberofVariables=1" + "\n")
fileObject26.write("#IntervalSize=" + str(max_iter) + "\n")
fileObject26.write("TitleText: OpenModelica simulation plot" + "\n")
fileObject26.write("XLabel: Frequency" + "\n")
fileObject26.write("DataSet: Param_var" + "\n")
fileObject26.close()
# ----------------------------------------------------------------------------
# Run OpenModelica Script
# ----------------------------------------------------------------------------
for k in range(1,max_iter+1):
   jj=k
   sweepVariable = str(pow(10,jj))
   execfile('make_OMscript.py')
   os.popen(r"C:\OpenModelica1.7.0\bin\omc.exe test.mos").read()
   #Retrieving and restoring the result
   execfile('sim.py')

os.system(r'"C:\OpenModelica1.7.0\bin\OMPlot.exe --title=Parametric_Sweep --
filename=ParametricSweep.plt --legend=true --grid=true --logx=false --
logy=false --xlabel=Frequency --ylabel=Source_vAC --plot Param_var"')
import time
```

```
now2 = time.time()
print 'Time elapsed: '+ str(now2-now1)
```

sim.py

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%               File name: sim.py
#%               last modified 2011-07-06 david
#%               2011 - David Malo Cid
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#
# The script extract data from simulation result.
#
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# load modules
# ---------------------------------------------------------------------------
#import sys, os
# Function to create a vector of zeros.
# ---------------------------------------------------------------------------
def zeros(n):
     vec = [0.0]
     for i in range(int(n)-1):
            vec = vec + [0.0]
     return vec
# Open OpenModelica result file for reading
# ---------------------------------------------------------------------------
fileObject5=open("test_res.plt",'r',1)
# Read simulation interval size
# ---------------------------------------------------------------------------
line = fileObject5.readline()
size = int(fileObject5.readline().split('=')[1])
#print size
time = zeros(size)
source_vAC_mag_dB = zeros(size)
while line != ['DataSet: time\n']:
     line = fileObject5.readline().split(',')[0:1]
for j in range(int(size)):
     time[j] = float(fileObject5.readline().split(',')[0]) #[0:2]
while line != ['DataSet: nmostransistor1.source_vAC_mag_dB\n']:
     line = fileObject5.readline().split(',')[0:1]
for j in range(int(size)):
     source_vAC_mag_dB[j] =float(fileObject5.readline().split(',')[1])
# Close OpenModelica result file
# ---------------------------------------------------------------------------
fileObject5.close()
# Open simulation output file for writing
# ---------------------------------------------------------------------------
fileObject26=open('ParametricSweep.plt','a+',1)
# ---------------------------------------------------------------------------
fileObject26.write(str(sweepVariable)+ ", " + str(source_vAC_mag_dB[1]) +
"\n")
fileObject26.close()
```

make_OMScript.py

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%               File name: make_OMScript.py
#%               last modified 2011-07-06 david
#%               2011 - David Malo Cid
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#
# This script makes the *.mos file. The *.mos file can then be used to run a
# (Open)Modelica model as script.
#
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# load modules
# ---------------------------------------------------------------------------
#import sys, os, string
# (Open) Write the OpenModelica Script file
# ---------------------------------------------------------------------------
fileObject2=open('test.mos','w',1)
fileObject2.write("loadModel(Modelica);\n")
fileObject2.write("loadFile(\"test.mo\");\n")
fileObject2.write("setComponentModifierValue(test,freq,Code(="+str
(sweepVariable)+"));\n")
#Simulate and plot
# ---------------------------------------------------------------------------
fileObject2.write("simulate(test,
stopTime=0.00001,tolerance=0.00001,outputFormat=\"plt\");\n")
# Close OpenModelica Script file
# ---------------------------------------------------------------------------
fileObject2.close()
```