SystemC-WMS:Wave Mixed Signal Simulator

Simone Orcioni, Giorgio Biagetti, and Massimo Conti

***Abstract*— This work presents a methodology for extending SystemC to mixed signal heterogeneous systems. To this end, a method for modelling analog modules using wave quantities is proposed, and a new kind of port and channel have been defined. This class library is plugged directly on top of the standard SystemC kernel, so as to allow a seamless integration with the pre-existing simulation environment.**

I. Introduction

SYSTEMC is an emerging tool for the description and simulation of hardware and software at system level [1], and it is not rare that this high level of abstraction could require the interfacing of both digital and analog parts. To this aim it has been proposed [2] to constitute an Open SystemC Initiative (OSCI) Working Group devoted to the development of an extension of SystemC to mixed-signal simulation: SystemC-AMS.

The basic SystemC methodology [1] makes use of modules and interfaces to describe complex systems. Modules communicate through interfaces, implemented in a channel, by calling methods in the channel. Conversely, events in the channel can activate modules connected to the channel itself. The present work proposes a methodology for the description of analog blocks using only such instruments and libraries. Taking advantage of this communication scheme and of the underlying SystemC kernel, we implement the various analog parts of a system as analog modules, which communicate by exchanging energy waves through wavechannel interfaces.

II. Modelling of Analog Modules in SystemC

The main goal of our methodology is to describe each analog module, which comprises a system, by means of a system of nonlinear ordinary differential equations (ODE) of the following type:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \end{cases} \tag{1}$$

The authors are with the Dipartimento di Elettronica, Intelligenza artificiale e Telecomunicazioni, Università Politecnica delle Marche, I-60131 Ancona, Italy (e-mail: SystemC-WMS@deit.univpm.it).

Publisher Item Identifier ...

Fig. 1. On the left, electrical port symbol using wave quantities

where $\boldsymbol{f}$ and $\boldsymbol{g}$ are vector expressions describing system dynamics, while $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{u}$ are state, output and input vectors, respectively. Equation (1) should describe a part of the system under consideration, like an N-port modelled at circuital level, or it may represent a high level macromodel describing the part functionality.

A. Module representation based on wave-exchange

The use of the incident/reflected wave model [3] greatly simplify the problems of taking into account the connection between modules, since it can be mandated that modules use incident waves as inputs and produce reflected waves as outputs.

Our approach uses, like in the WDF theory, the $a\,b$ parameters as input/output signals and implements the duties of the scattering junction in a new entity called wavechannel, complying with SystemC conventions for channels.

To explain its functionality, without loss of generality we can fix our attention to an N-port in the electrical domain, described through its port quantities v_j and i_j, $j = 1, \ldots, N$.The relation between electrical and wave quantities can be obtained from the following definition of incident (a_j) and reflected (b_j) wave:

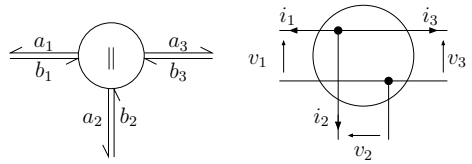
$$\begin{aligned} a_j &= \tfrac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \tfrac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned} \tag{2}$$

so that $a_j^2 - b_j^2$ is the instantaneous power entering port j and R_j is a normalization resistance. Similar relationships hold for other domains as well.

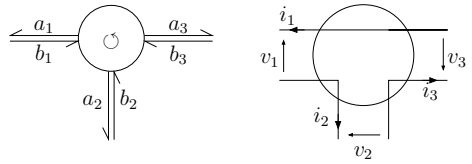
B. Wavechannels

Wavechannels are the means by which modules described by wave quantities communicate.

Consider a junction between N ports, each with its own normalization factor R_j. Let $\boldsymbol{v}$ and $\boldsymbol{i}$ be the voltage



(a) parallel interconnection



(b) series interconnection

Fig. 2. Wavechannel symbols corresponding to different port interconnections.

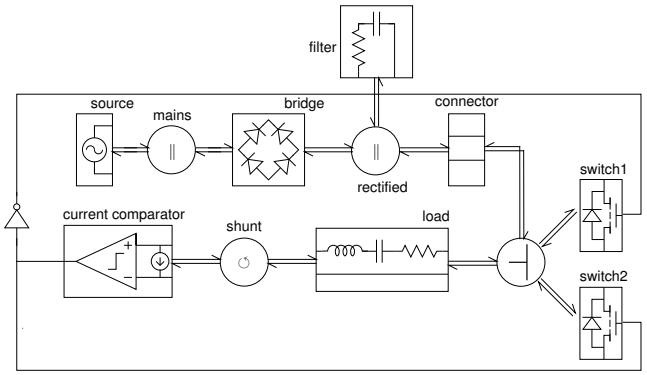


Fig. 3. Self-oscillating half-bridge inverter: SystemC modules and wavechannels interconnection diagram.

and current vector, respectively, and:

$$\begin{cases} \boldsymbol{A_v}\,\boldsymbol{v} = \boldsymbol{0} \\ \boldsymbol{A_i}\,\boldsymbol{i} = \boldsymbol{0} \end{cases} \tag{3}$$

be a complete and minimal set of Kirchhoff's equations describing the junction ($[\boldsymbol{A_v}]_{ij}, [\boldsymbol{A_i}]_{ij} \in \{0, \pm 1\}$). We maintain that letting:

$$\boldsymbol{A_x} = \boldsymbol{A_v} \operatorname*{diag}_{k=1,\ldots,N} R_k \quad \text{and} \quad \boldsymbol{A_y} = \boldsymbol{A_i} \operatorname*{diag}_{k=1,\ldots,N} 1/R_k \tag{4}$$

the scattering matrix $\boldsymbol{S}$ (such that $\boldsymbol{a} = \boldsymbol{S}\,\boldsymbol{b}$), by substituting the inverse of (2) and (4) into (3), becomes:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{A_x} \\ \boldsymbol{A_y} \end{bmatrix}^{-1} \begin{bmatrix} -\boldsymbol{A_x} \\ \boldsymbol{A_y} \end{bmatrix} \tag{5}$$

where $\boldsymbol{b}$ are the waves reflected by modules and thus entering the junction, whence $\boldsymbol{a}$ are scattered back from the junction to the modules thereby interconnected.

The above formulation can be used for any kind of junction.

C. SystemC class library

To let the implementation of systems by means of analog blocks described with wave quantities, a number of templates and classes have been designed: a new kind of port to let modules communicate via wave quantities (`ab_port`), the wavechannel that can interconnect them and that does the real computation of the scattering that occures at junctions (`ab_signal`), and a template base class (`wave_module`) that extends `analog_module` taking care of handling sensitivity lists and port declarations.

Ports expose an interface that allows users to read the incident wave value and to report (`write`) the reflected wave value. Of course, in the same `wave_module`, they can freely be mixed with standard SystemC ports and, in the same design, istantaneous analog modules, SFG analog modules (`analog_module`), and wave modules (`wave_module`) can be used.

To ease the implementation of complex systems with SysytemC-WMS, a number of templates and classes of linear and nonlinear modules have been provided.

III. Application examples

As an example of a possible application of SystemC-WMS to a real problem a self-oscillating half-bridge (HB) inverter has been chosen. The self-oscillating HB is made up of a half-bridge provided with positive feedback carried out by sensing the load current with a current comparator. The inverter is used to delivery power from a DC source to an AC load, constituted by an RLC series.

The main components of the circuit are the switches, the Graetz' bridge used to rectify the line voltage, and the voltage source used to convert stimuli from a standard SystemC signal or an SFG representation to the wave representation. All the analog modules are connected together by means of wavechannels, as shown in Fig. 3.

The switch module models the behavior of an ideal switch, like an MOS switch with zero on resistance, coupled in parallel with a bypass ideal diode, and with an additional logical input to control the switch.

The full library and other application examples, are available from the SystemC-WMS web site [4].

A. Simulation results

The circuits has been simulated using the SystemC-WMS, using a fourth-order Adams-Moulton predictor-corrector ODE solver, taking 4.064s on a 1 GHz Pentium-M computer. In particular the self oscillating bridge has been simulated during power-on functioning. After a $200\,\mu$s of transient, an abruput change from $80\ \mu$H to $40\ \mu$H of the inductance of the load has been imposed to simulate a load change.

Results are shown in Figs. 4, where the load current is plotted as a function of time.

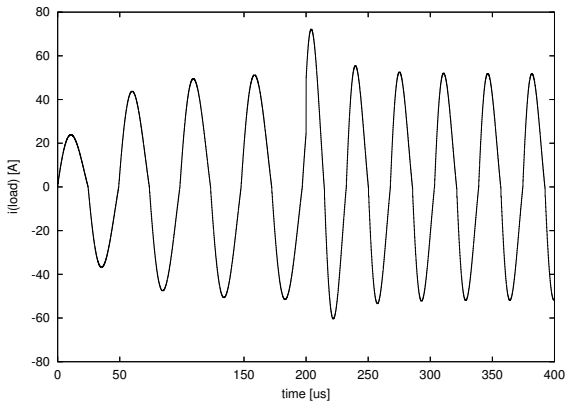


Fig. 4. Simulation of the current into the load.

IV. Conclusion

The increasing complexity of systems and circuits ask for an easy way to model and simulate the overall behavior of a complex system spanning multiple domains. In order for SystemC to be able to cope with these requirements, an extension aimed at allowing the modeling and simulation of analog circuits is mandatory.

This work proposes an effective, and still not excessively complex framework, that simplify the modeling of the interaction between analog models belonging to heterogeneous domains, as well as model reuse. By using power waves as standard input/output signals for analog modules, these can be independently modeled and freely interconnected together in arbitrary topologies without having to deal with complex interface compatibility issues. Moreover, this allows for a uniform treatment of heterogeneous domains, thus paving the way to the development of truly generic and reusable model libraries.

References

[1] The Open SystemC Initiative, (OSCI), "SystemC documentation," online, http://www.systemc.org.

[2] K. Einwich, "Analog mixed signal extensions for SystemC," http://mixsigc.eas.iis.fhg.de, White paper and proposal for the foundation of the SystemC-AMS OSCI Working Group, 2002.

[3] K. Kurokawa, "Power waves and the scattering matrix," *IEEE Trans. Microwave Theory Tech.*, vol. 13, no. 2, pp. 194–202, Mar. 1965.

[4] G. Biagetti, M. Conti, and S. Orcioni, "SystemC-WMS home page," online, http://www.deit.univpm.it/systemc-wms/.

SystemC-WMS: A Wave Mixed Signal Simulator

Simone Orcioni
Giorgio Biagetti, Massimo Conti

DEIT, Dipartimento di Elettronica, Intelligenza artificiale e Telecomunicazioni
Università Politecnica delle Marche

C/C++-Based Modelling of Embedded Mixed-Signal Systems

Outline

State of the Art

Simulation of heterogeneous systems comprising analog parts:

SystemC-AMS (Analog & Mixed Signal)

- Modeling with block diagrams and directed signals
- Only linear systems can be modeled up to now
- Synchronous Data Flow MoC
- It is an OSCI effort

SystemC-WMS (Wave Mixed Signal)

- Wave theory-based "analog interfaces" used to interconnect blocks and automatically account for load effects
- Any locally-ODE nonlinear system can be modeled
- Automatic variable-time step ODE solver
- Fully compatible with SystemC-2.2.

State of the Art

Simulation of heterogeneous systems comprising analog parts:

SystemC-AMS (Analog & Mixed Signal)

- Modeling with block diagrams and directed signals
- Only linear systems can be modeled up to now
- Synchronous Data Flow MoC
- It is an OSCI effort

SystemC-WMS (Wave Mixed Signal)

- Wave theory-based "analog interfaces" used to interconnect blocks and automatically account for load effects
- Any locally-ODE nonlinear system can be modeled
- Automatic variable-time step ODE solver
- Fully compatible with SystemC-2.2.

State of the Art

Simulation of heterogeneous systems comprising analog parts:

SystemC-AMS (Analog & Mixed Signal)

- Modeling with block diagrams and directed signals
- Only linear systems can be modeled up to now
- Synchronous Data Flow MoC
- It is an OSCI effort

SystemC-WMS (Wave Mixed Signal)

- Wave theory-based "analog interfaces" used to interconnect blocks and automatically account for load effects
- Any locally-ODE nonlinear system can be modeled
- Automatic variable-time step ODE solver
- Fully compatible with SystemC-2.2.

State of the Art

Simulation of heterogeneous systems comprising analog parts:

SystemC-AMS (Analog & Mixed Signal)

- Modeling with block diagrams and directed signals
- Only linear systems can be modeled up to now
- Synchronous Data Flow MoC
- It is an OSCI effort

SystemC-WMS (Wave Mixed Signal)

- Wave theory-based "analog interfaces" used to interconnect blocks and automatically account for load effects
- Any locally-ODE nonlinear system can be modeled
- Automatic variable-time step ODE solver
- Fully compatible with SystemC-2.2.

State of the Art

Simulation of heterogeneous systems comprising analog parts:

SystemC-AMS (Analog & Mixed Signal)

- Modeling with block diagrams and directed signals
- Only linear systems can be modeled up to now
- Synchronous Data Flow MoC
- It is an OSCI effort

SystemC-WMS (Wave Mixed Signal)

- Wave theory-based "analog interfaces" used to interconnect blocks and automatically account for load effects
- Any locally-ODE nonlinear system can be modeled
- Automatic variable-time step ODE solver
- Fully compatible with SystemC-2.2.

Outline

High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

- $\mathbf{x}$ state
- $\mathbf{u}$ input
- $\mathbf{y}$ output

High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

- **x** state
- **u** input
- **y** output

This description is not valid for

- DAE systems
- conservative-law systems

- Most systems can be described as locally ODE, globally DAE

High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

- **x** state
- **u** input
- **y** output

This description is not valid for

- DAE systems
- conservative-law systems

- Most systems can be described as locally ODE, globally DAE

High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

- **x** state
- **u** input
- **y** output

This description is not valid for

- DAE systems
- conservative-law systems

- Most systems can be described as locally ODE, globally DAE

High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

- $\mathbf{x}$ state
- $\mathbf{u}$ input
- $\mathbf{y}$ output

This description is not valid for

- DAE systems
- conservative-law systems

- Most systems can be described as locally ODE, globally DAE

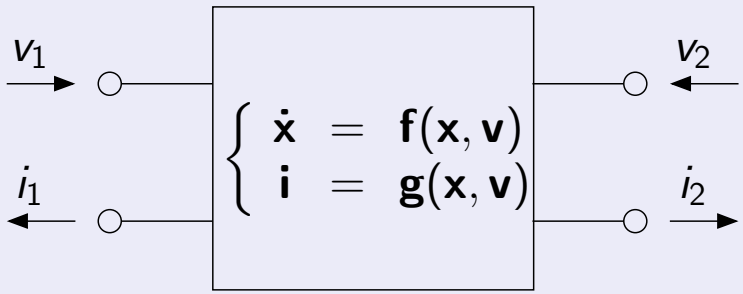
High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) \end{cases}$$

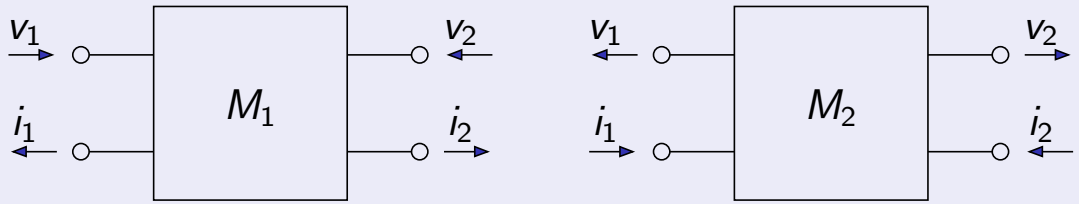
- **x** state
- **u** input
- **y** output

Modelling of a 2-port with voltage and current as input/output signals



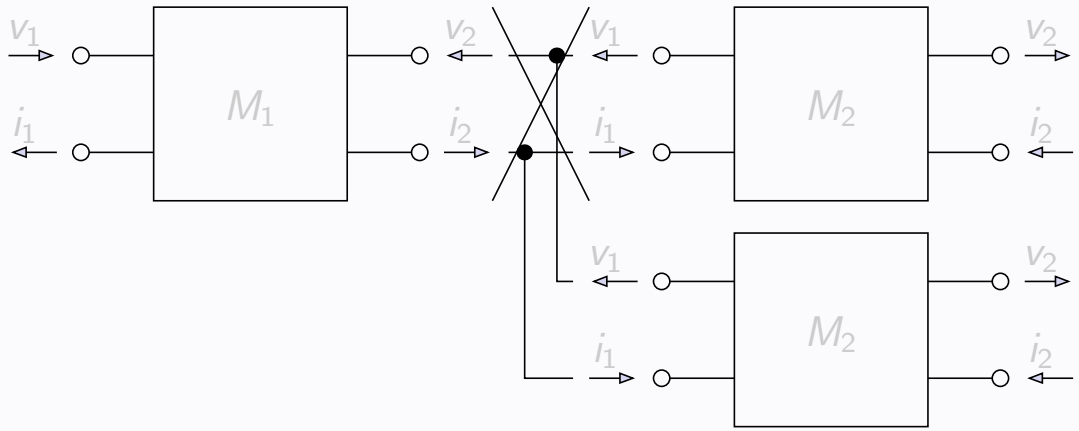
Modelling of Analog Modules Using Voltage and Current

Two different modules that can be cascaded



- In non-SFG no physical clue helps in choosing input/output quantities

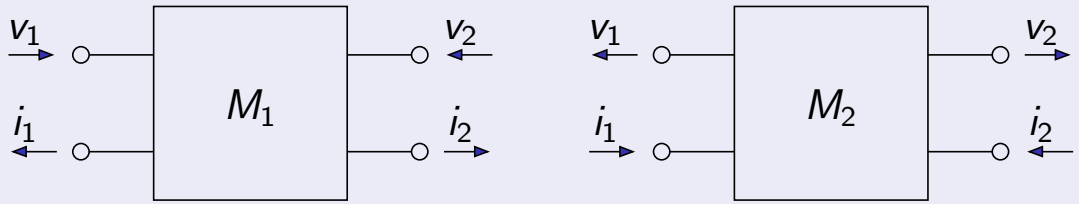
It would not be possible to connect them in series, parallel, or cascade them



- Some complex adaptor is needed to tie together the output voltages

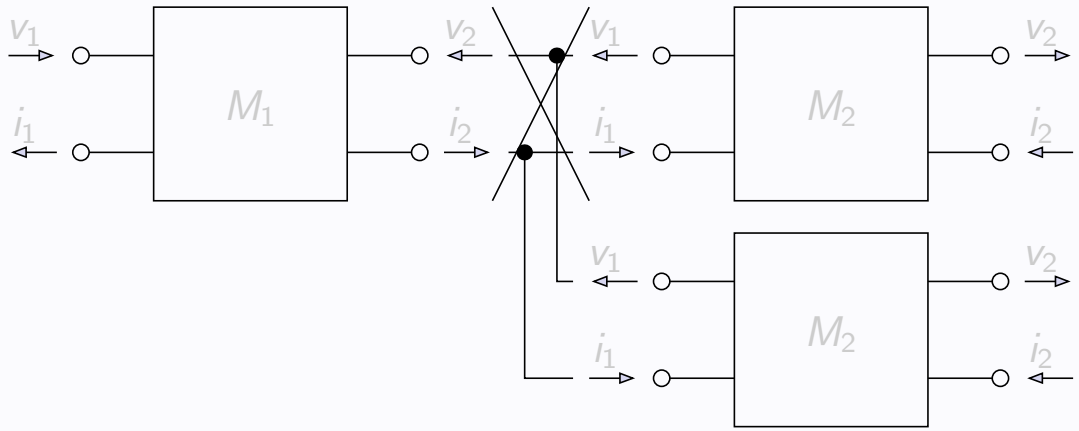
Modelling of Analog Modules Using Voltage and Current

Two different modules that can be cascaded



- In non-SFG no physical clue helps in choosing input/output quantities

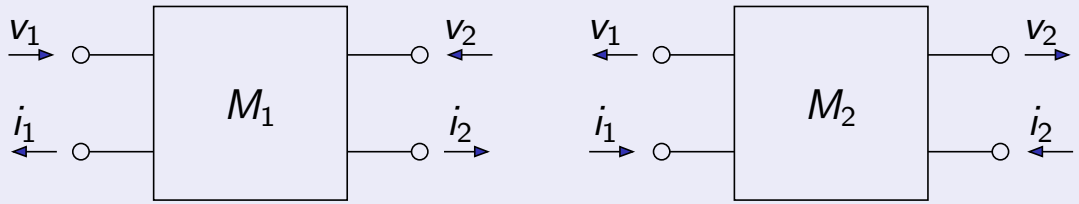
It would not be possible to connect them in series, parallel, or cascade them



- Some complex adaptor is needed to tie together the output voltages

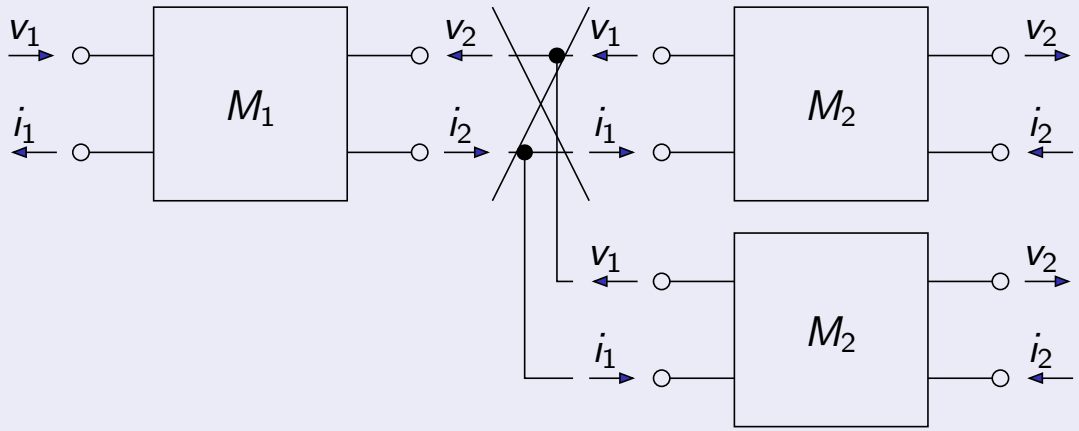
Modelling of Analog Modules Using Voltage and Current

Two different modules that can be cascaded



- In non-SFG no physical clue helps in choosing input/output quantities

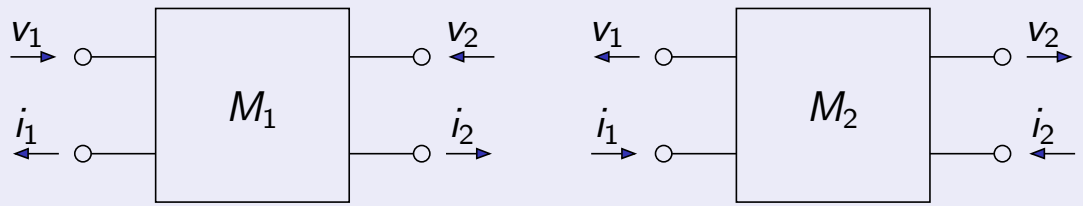
It would not be possible to connect them in series, parallel, or cascade them



- Some complex adaptor is needed to tie together the output voltages

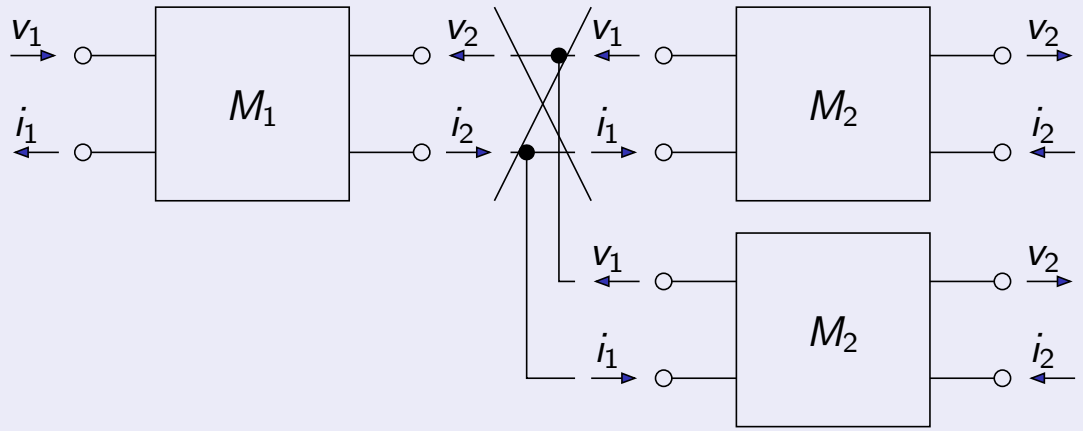
Modelling of Analog Modules Using Voltage and Current

Two different modules that can be cascaded



- In non-SFG no physical clue helps in choosing input/output quantities

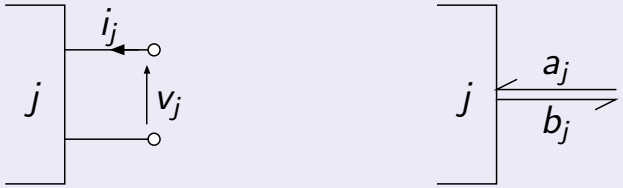
It would not be possible to connect them in series, parallel, or cascade them



- Some complex adaptor is needed to tie together the output voltages

a, b Parameter Modelling

Definition (Incident a_j and Reflected b_j wave)



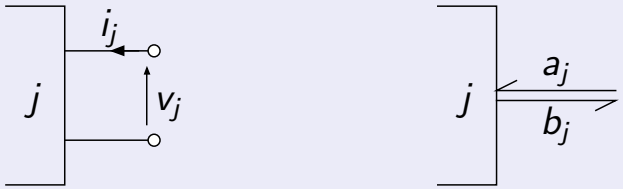
$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned}$$

- $a_j^2 - b_j^2$ is the instantaneous power entering port j
- R_j is a normalization resistance

- Models can easily be derived from analogous models based on voltage and current;
- Every device that possess a scatter-matrix representation can be modeled in this way

a, b Parameter Modelling

Definition (Incident a_j and Reflected b_j wave)



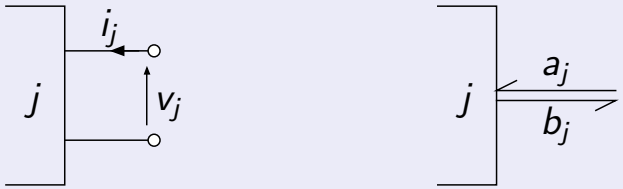
$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned}$$

- $a_j^2 - b_j^2$ is the instantaneous power entering port j
- R_j is a normalization resistance

- Models can easily be derived from analogous models based on voltage and current;
- Every device that possess a scatter-matrix representation can be modeled in this way

a, b Parameter Modelling

Definition (Incident a_j and Reflected b_j wave)



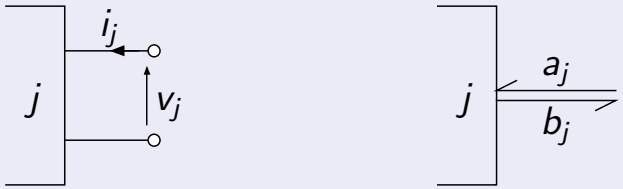
$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned}$$

- $a_j^2 - b_j^2$ is the instantaneous power entering port j
- R_j is a normalization resistance

- Models can easily be derived from analogous models based on voltage and current;
- Every device that possess a scatter-matrix representation can be modeled in this way

a, b Parameter Modelling

Definition (Incident a_j and Reflected b_j wave)



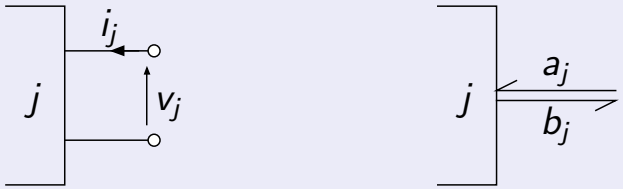
$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned}$$

- $a_j^2 - b_j^2$ is the instantaneous power entering port j
- R_j is a normalization resistance

- Models can easily be derived from analogous models based on voltage and current;
- Every device that possess a scatter-matrix representation can be modeled in this way

a, b Parameter Modelling

Definition (Incident a_j and Reflected b_j wave)



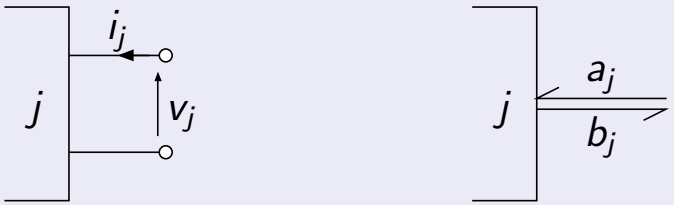
$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned}$$

- $a_j^2 - b_j^2$ is the instantaneous power entering port j
- R_j is a normalization resistance

- Models can easily be derived from analogous models based on voltage and current;
- Every device that possess a scatter-matrix representation can be modeled in this way

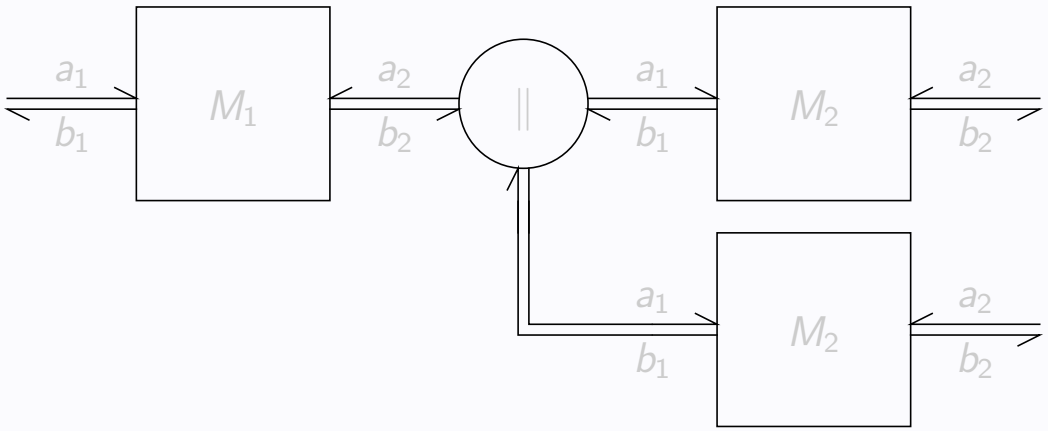
Modelling of Analog Modules Using Wave Quantities

Electrical port representation using wave quantities



- a is always the input and b the output of a port

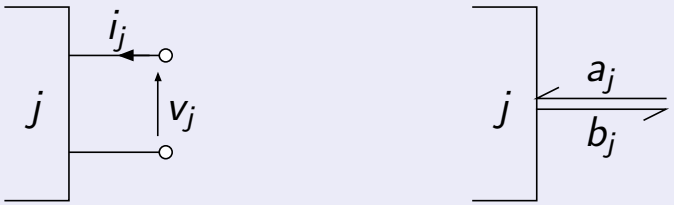
Solution to the interconnection problem by a wave adaptor



A simple and generic wave adaptor can dispatch waves to the modules thereby interconnected

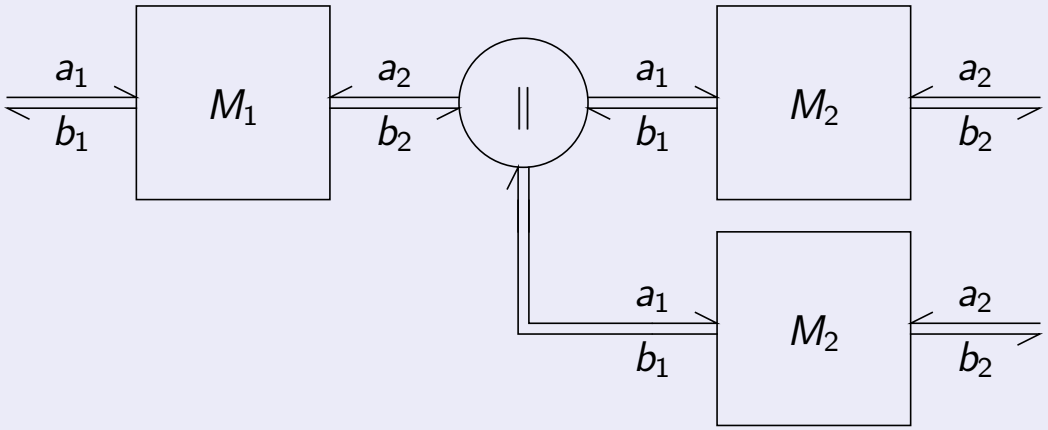
Modelling of Analog Modules Using Wave Quantities

Electrical port representation using wave quantities



- a is always the input and b the output of a port

Solution to the interconnection problem by a wave adaptor

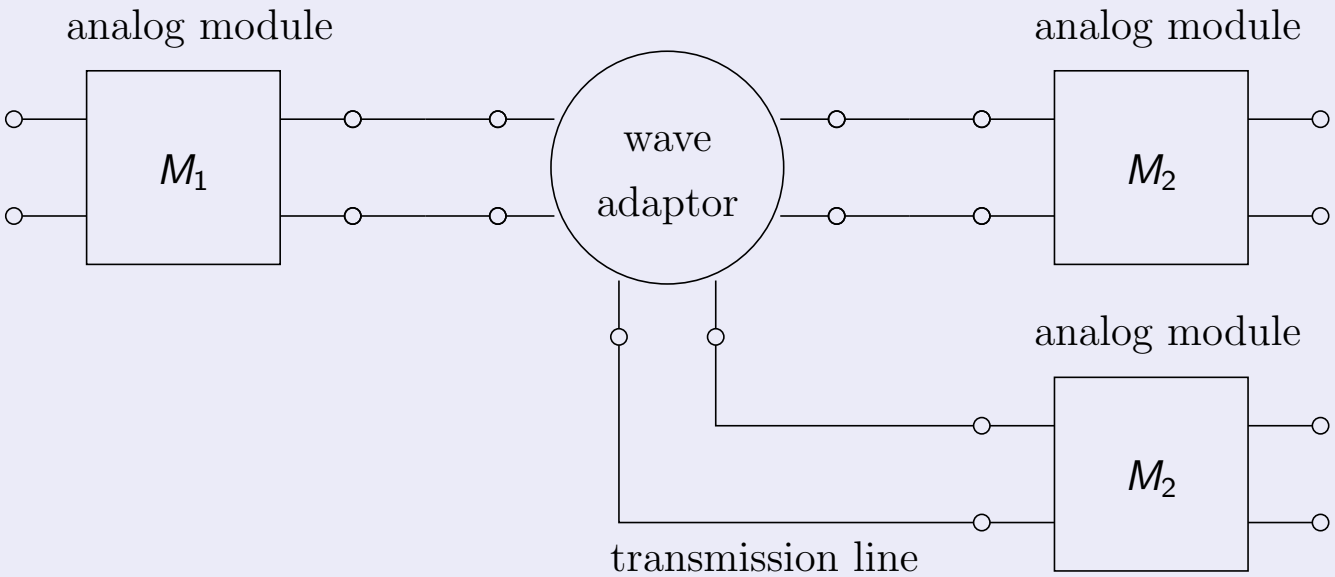


A simple and generic wave adaptor can dispatch waves to the modules thereby interconnected

Outline

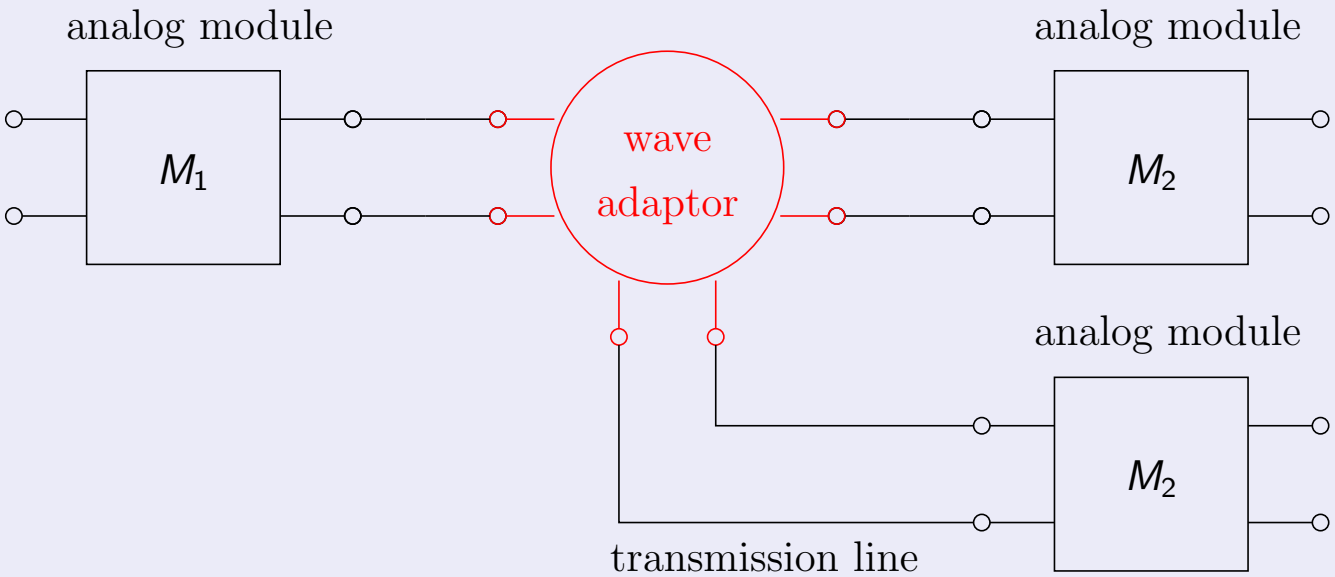
Wavechannel

Electrical schematic diagram



Wavechannel

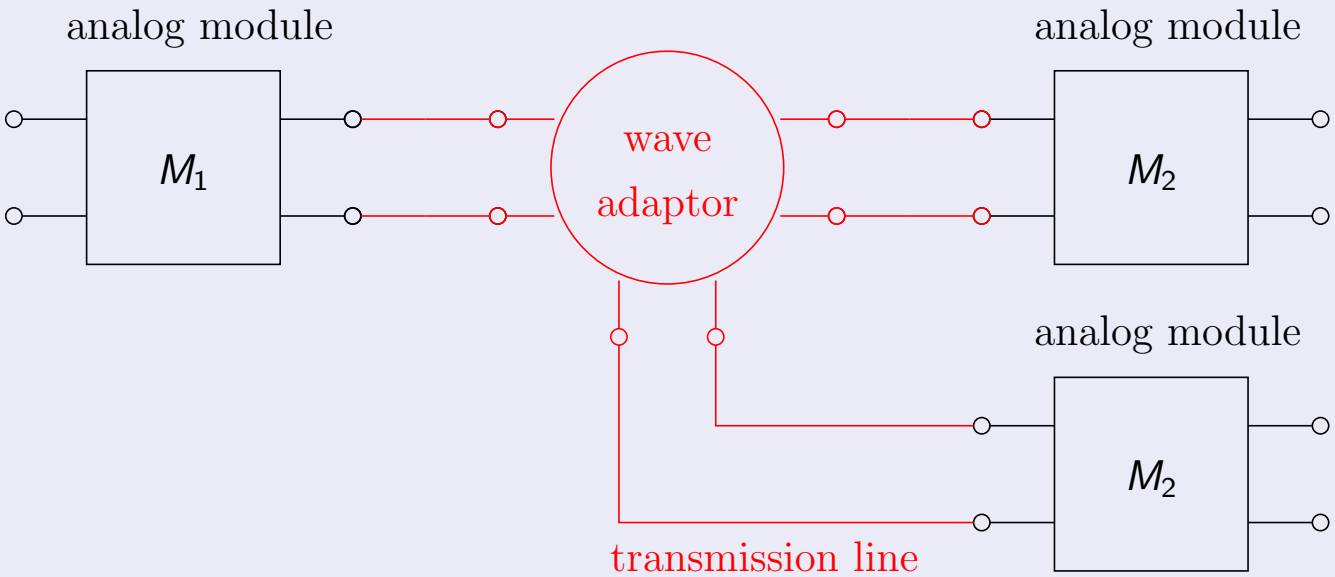
Electrical schematic diagram



A wavechannel can be represented by a junction box connected to ports by transmission lines

Wavechannel

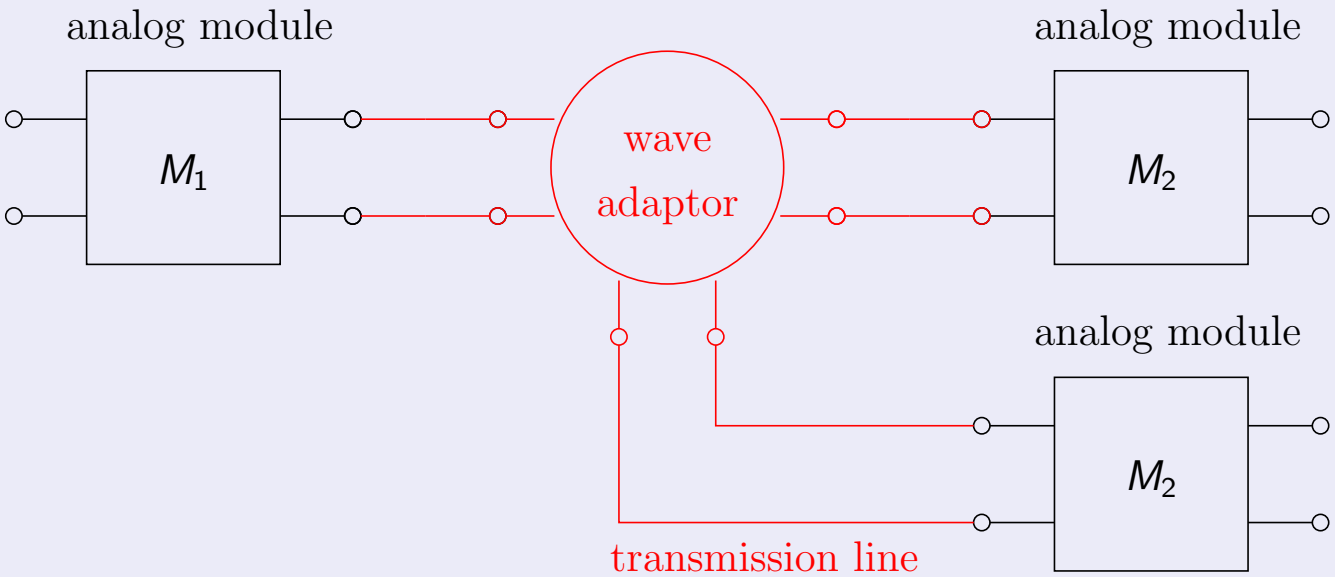
Electrical schematic diagram



A wavechannel can be represented by a junction box connected to ports by transmission lines

Wavechannel

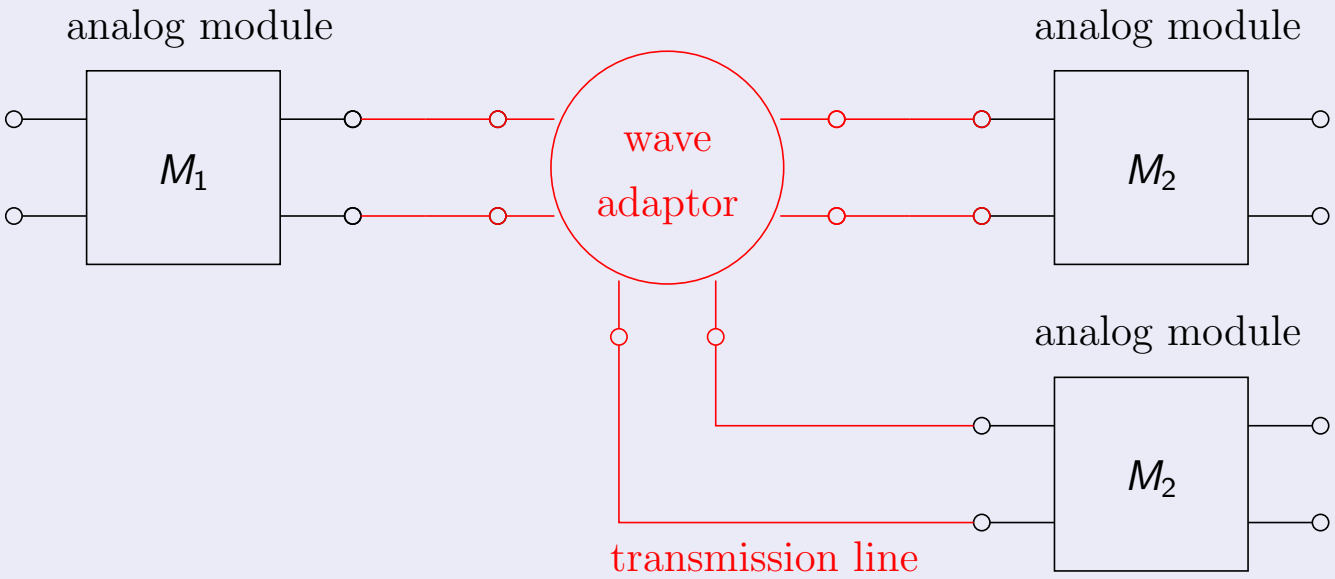
Electrical schematic diagram



Propagation delay can be excluded $\Rightarrow$ Delay-free loops $\Rightarrow$ locally ODE, globally DAE system

Wavechannel

Electrical schematic diagram



Fixed-point solution of DAE $\Leftrightarrow$ Solution of Maxwell's equations in quasi-static conditions $\Leftrightarrow$ Kirchhoff's equations

Wavechannels

Consider a junction between N ports

Let $\mathbf{v}$ and $\mathbf{i}$ be the voltage and current vector, respectively, and:

$$\begin{cases} \mathbf{A_v}\,\mathbf{v} = \mathbf{0} \\ \mathbf{A_i}\,\mathbf{i} = \mathbf{0} \end{cases}$$

be a complete and minimal set of Kirchhoff's equations describing the junction, where $[\mathbf{A_v}]_{ij}$ and $[\mathbf{A_i}]_{ij}$ are matrices of 0 and ± 1. Letting:

$$\mathbf{A_x} = \mathbf{A_v} \underset{k=1,\ldots,N}{\operatorname{diag}} R_k \quad \text{and} \quad \mathbf{A_y} = \mathbf{A_i} \underset{k=1,\ldots,N}{\operatorname{diag}} 1/R_k$$

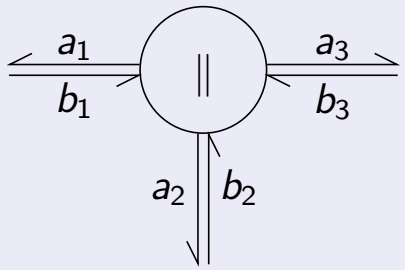
the scattering matrix $\mathbf{S}$ (such that $\mathbf{a} = \mathbf{S}\,\mathbf{b}$), becomes:

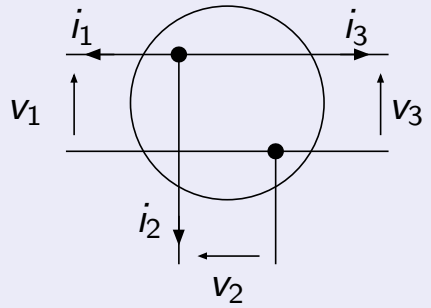
$$\mathbf{S} = \begin{bmatrix} \mathbf{A_x} \\ \mathbf{A_y} \end{bmatrix}^{-1} \begin{bmatrix} -\mathbf{A_x} \\ \mathbf{A_y} \end{bmatrix}$$

Parallel wavechannel

$$\sum_{j=1}^{N} i_j = 0$$

$$v_1 = v_2 = \cdots = v_N$$





which leads to:

$$\mathbf{A_v} = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots \\ & & \vdots & & \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix}$$

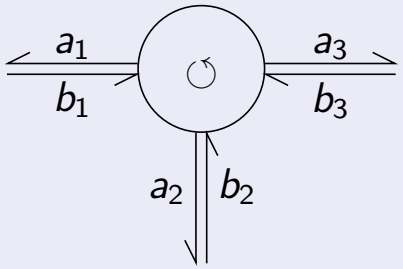
$$\mathbf{A_i} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

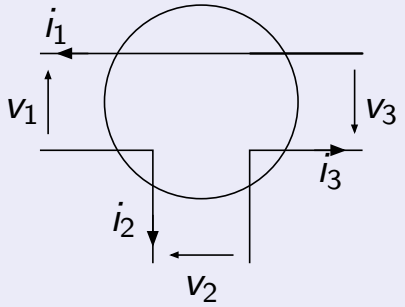
- $N = 1 \quad \Rightarrow \quad$ open circuit

Series wavechannel

$$\sum_{j=1}^{N} v_j = 0$$

$$i_1 = i_2 = \cdots = i_N$$





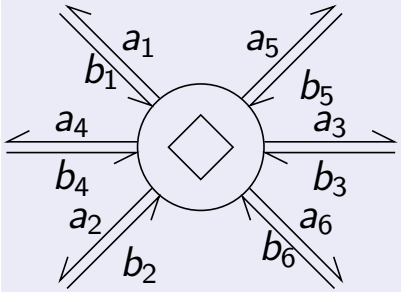
which leads to:

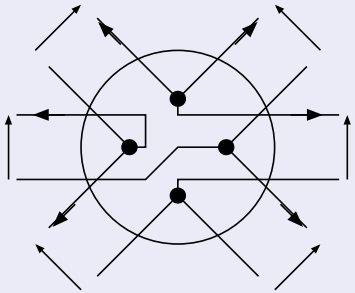
$$\mathbf{A_v} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

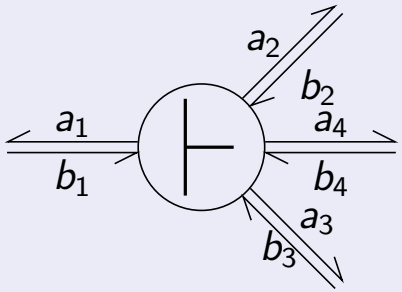
$$\mathbf{A_i} = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots \\ & & \vdots & & \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix}$$

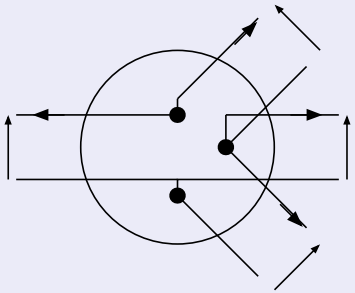
- $N = 1 \quad \Rightarrow \quad$ shunt

Bridge — Half-bridge









$$\mathbf{A_v} = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 \\ 1 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}$$

$$\mathbf{A_i} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

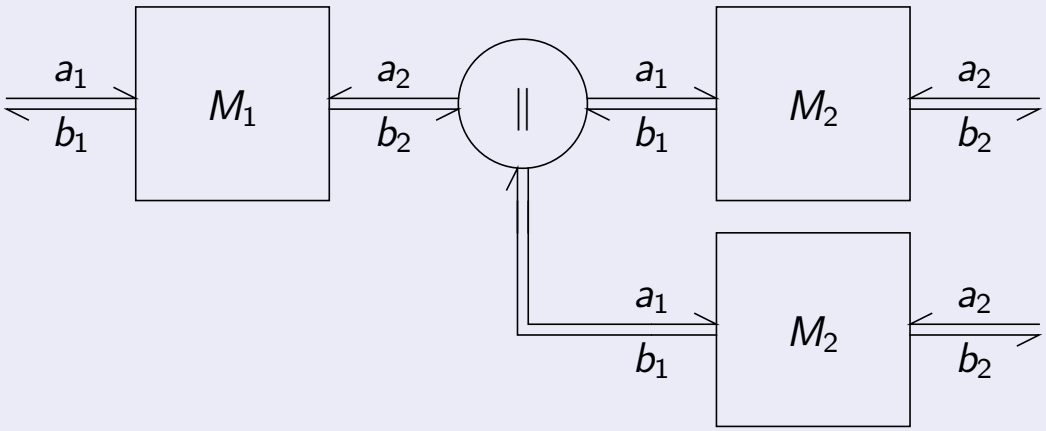
$$\mathbf{A_v} = \begin{bmatrix} 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

$$\mathbf{A_i} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

Outline

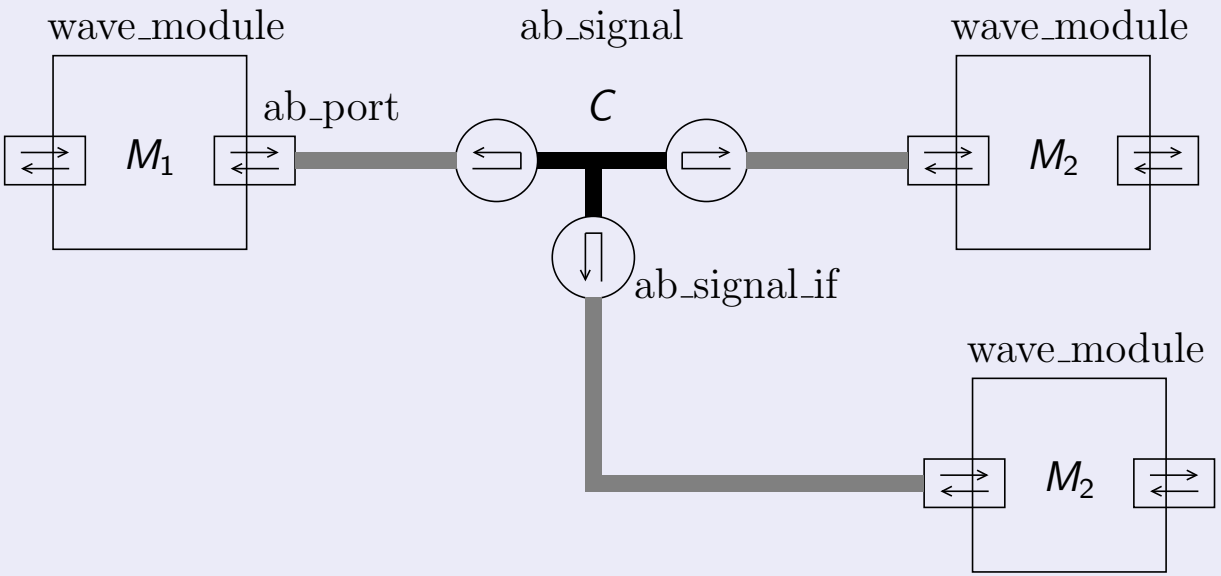
SystemC-WMS class library

a,b parameter modelling diagram



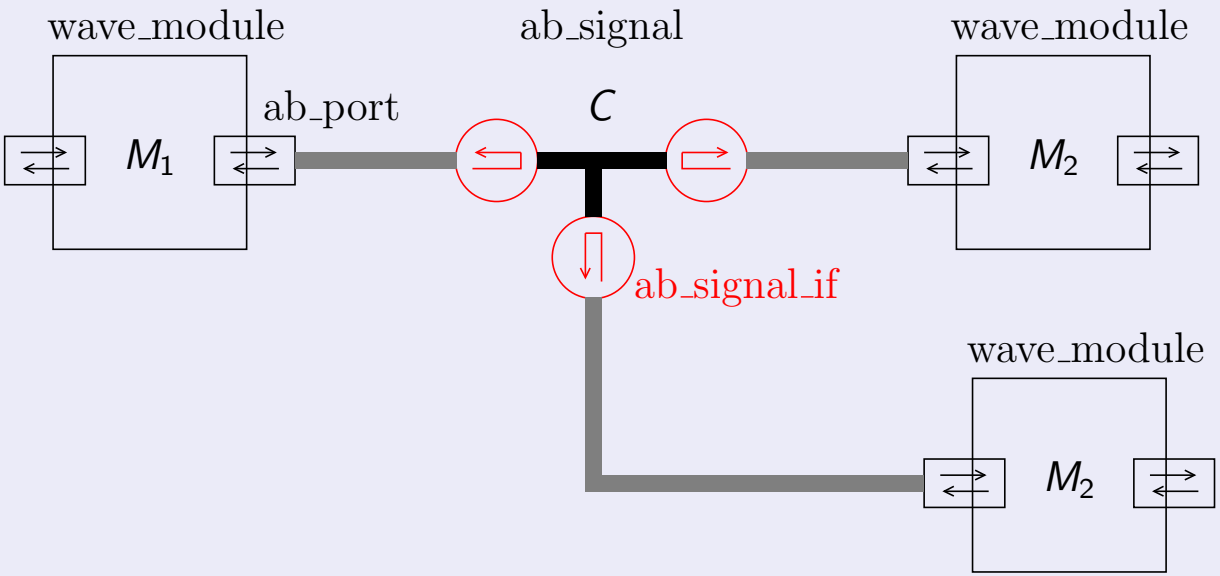
SystemC-WMS class library

SystemC-WMS class mapping



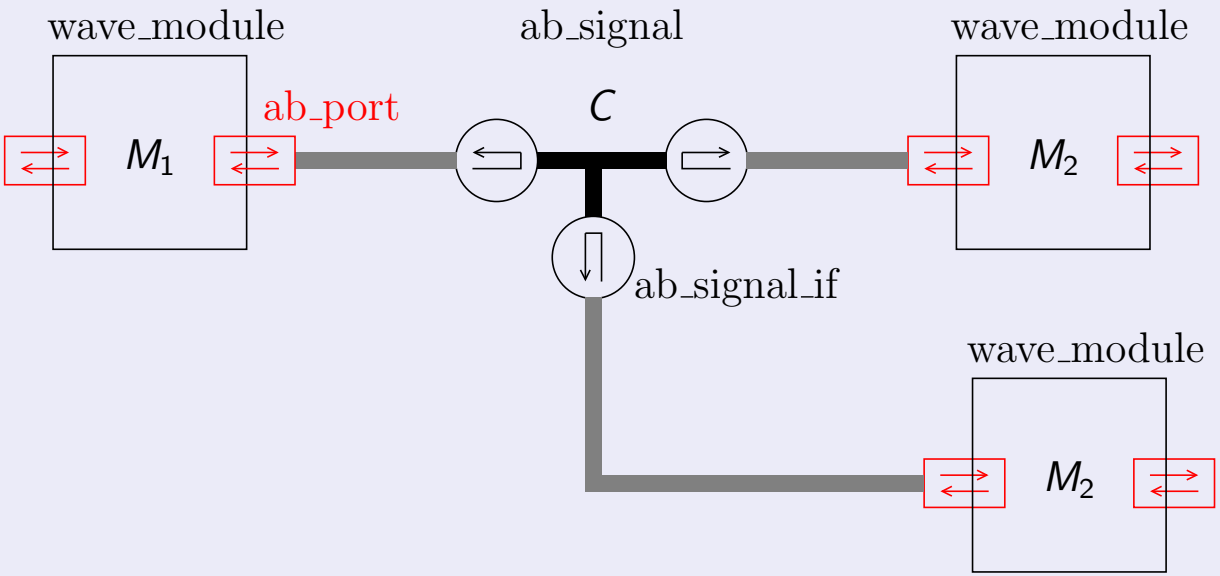
SystemC-WMS class library

SystemC-WMS class mapping



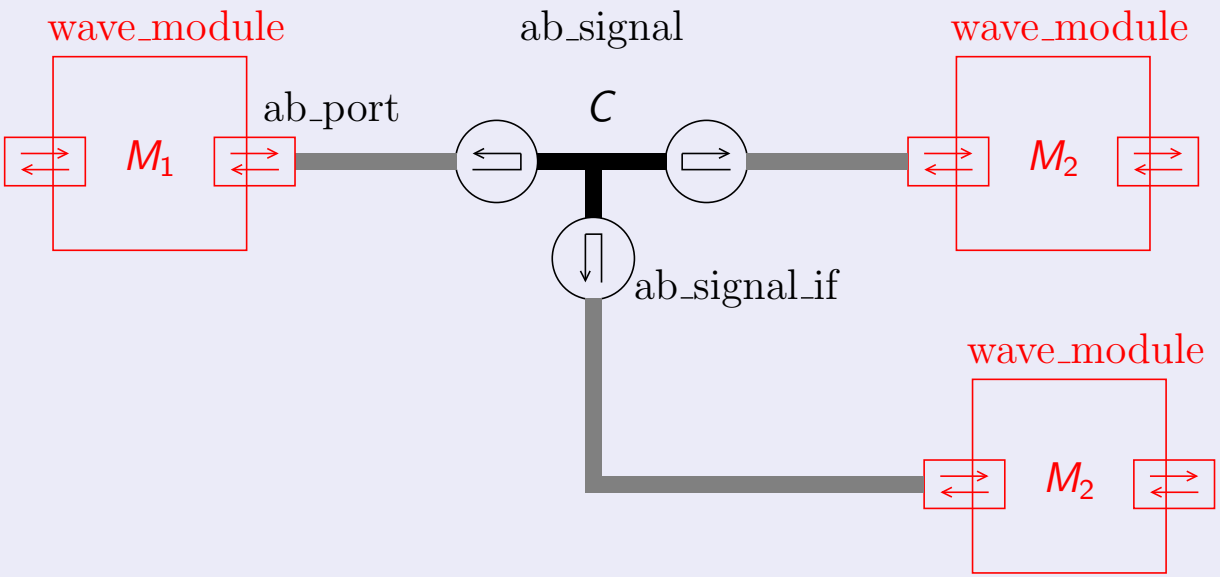
SystemC-WMS class library

SystemC-WMS class mapping



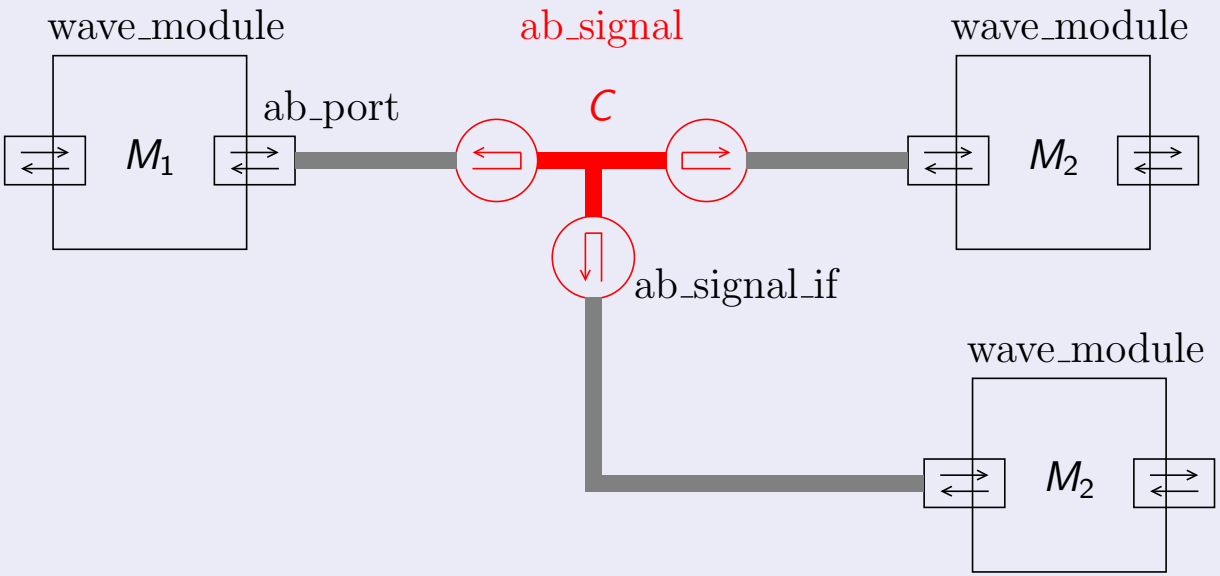
SystemC-WMS class library

SystemC-WMS class mapping



SystemC-WMS class library

SystemC-WMS class mapping



Wave module

Example (wave_module)

```
struct example : wave_module <1, electrical>, analog_module
{
  // state variable x is inherited from analog_module
  void field (double *var) const;
  void calculus ();
  SC_CTOR(example) : analog_module(...)
  {
    SC_THREAD(calculus);
    sensitive << activation;
  }
};
```

Wave module

Example (wave_module)

```
struct example : wave_module <1, electrical>, analog_module
{
  // state variable x is inherited from analog_module
  void field (double *var) const;
  void calculus ();
  SC_CTOR(example) : analog_module(...)
  {
    SC_THREAD(calculus);
    sensitive << activation;
  }
};
```

Wave module

Example (wave_module)

```
struct example : wave_module <1, electrical>, analog_module
{
  // state variable x is inherited from analog_module
  void field (double *var) const;
  void calculus ();
  SC_CTOR(example) : analog_module(...)
  {
    SC_THREAD(calculus);
    sensitive << activation;
  }
};
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);            // evaluate state change
}

void example::calculus ()
{
  x = 0;                       // state initialization here
  while (step()) {             // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);        // compute reflected wave
    port->write(b);            // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read();  // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (wave_module)

```
void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}
```

Wave module

Example (nature)

```
struct electrical : nature <double>
{
        static const char *across ()  {return "voltage";}
        static const char *through () {return "current";}
};

struct mechanical : nature <double>
{
        static const char *across ()  {return "speed";}
        static const char *through () {return "force";}
};

struct acoustical : nature <double>
{
        static const char *across ()  {return "pressure";}
        static const char *through () {return "volume velocity";}
};
```

Computation

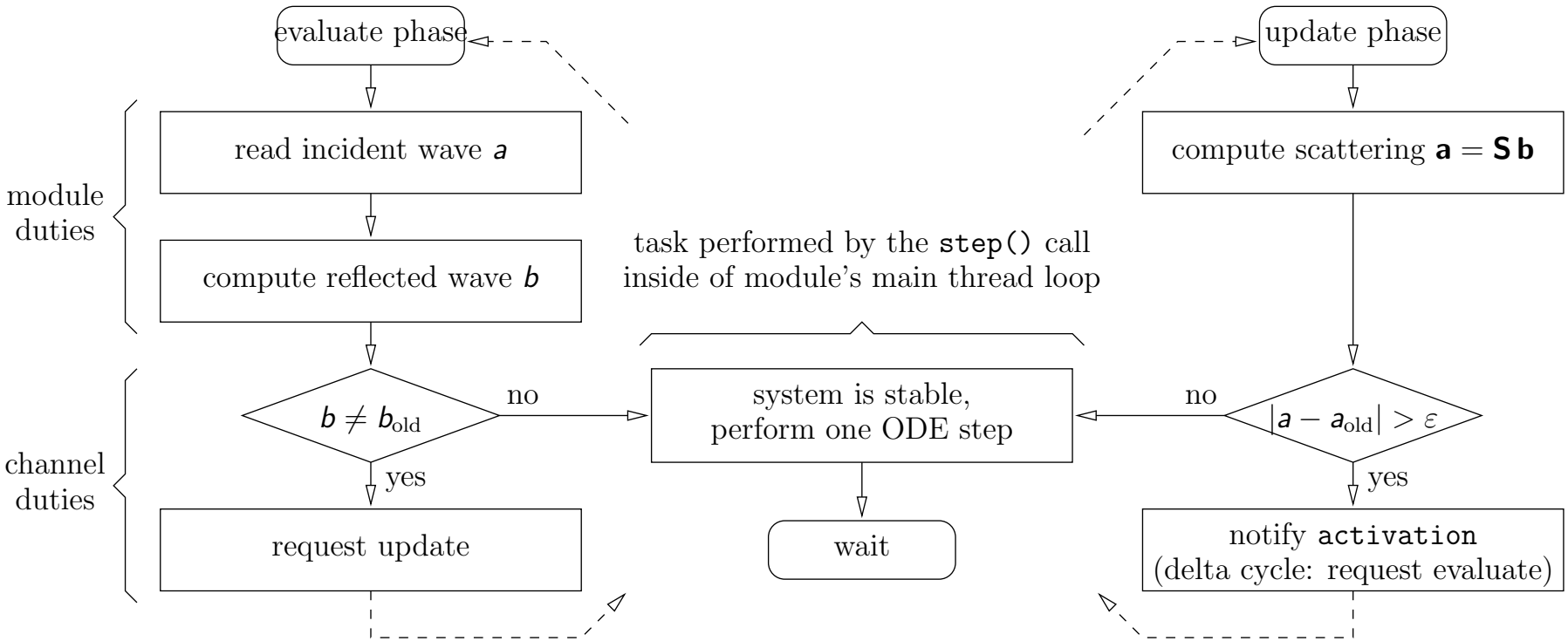
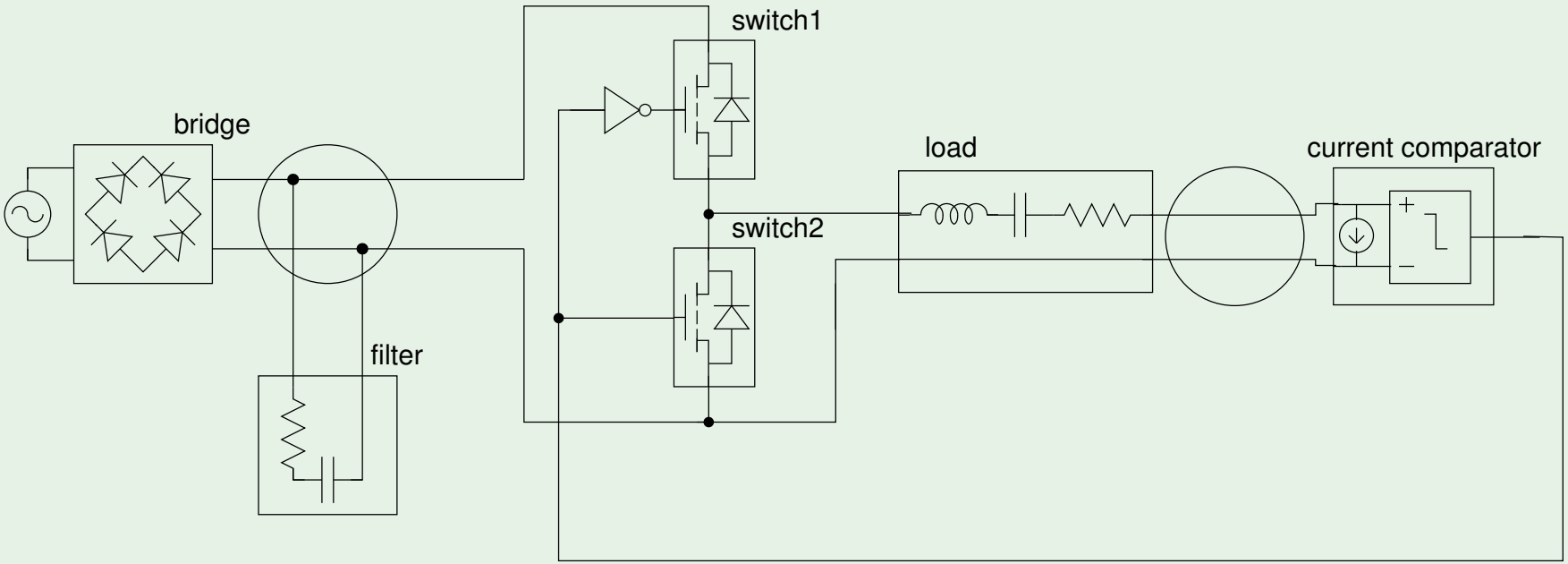


Figure: Sketch of the execution flow inside modules and wavechannels.

Outline

Self-oscillating half-bridge

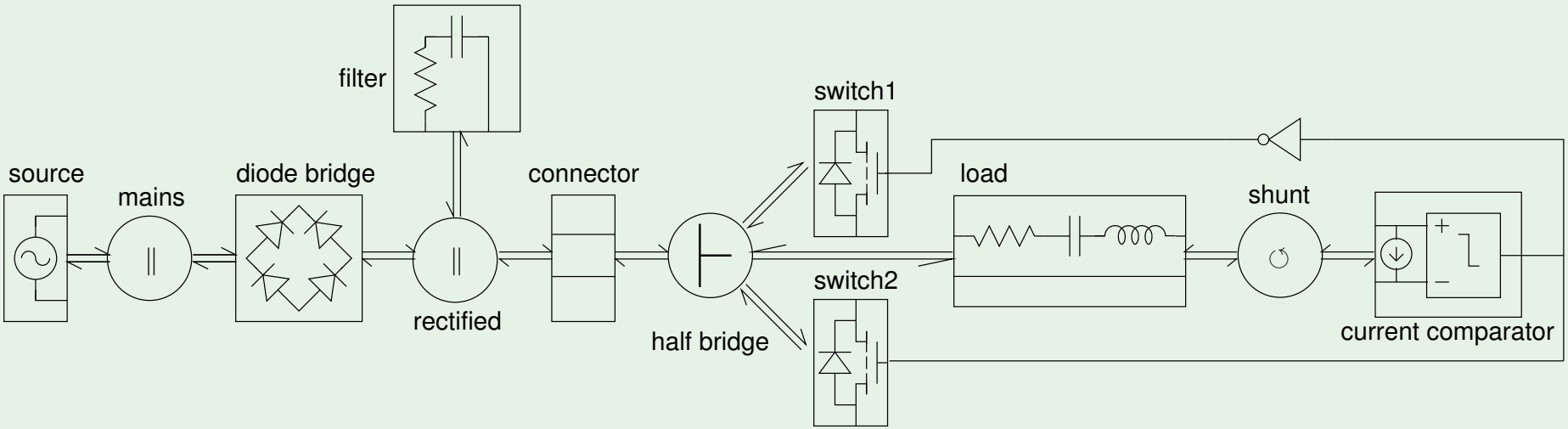
Example



Electrical schematic diagram

Self-oscillating half-bridge

Example



SystemC-WMS modules and wavechannels interconnection diagram

Self-oscillating half-bridge

Example

```
int sc_main (int argc, char *argv[])
{
  sc_core::sc_signal <bool> compout, pulse1, pulse2;
  sc_core::sc_signal <electrical::wave_type> in;

  ab_signal <electrical, parallel> mains, rectified;
  ab_signal <electrical, series> shunt;
  ab_signal <electrical, half_bridge> halfbridge;

  generator <electrical::wave_type> signal_source("SOURCE1",
                                  sine(sqrt(2) * 230 V, 50 Hz, pi/2));
  signal_source(in);

  source <electrical> wave_source("SOURCE2", cfg::across);
  wave_source(mains, in);
```

Self-oscillating half-bridge

Example

```
int sc_main (int argc, char *argv[])
{
  sc_core::sc_signal <bool> compout, pulse1, pulse2;
  sc_core::sc_signal <electrical::wave_type> in;

  ab_signal <electrical, parallel> mains, rectified;
  ab_signal <electrical, series> shunt;
  ab_signal <electrical, half_bridge> halfbridge;

  generator <electrical::wave_type> signal_source("SOURCE1",
                                  sine(sqrt(2) * 230 V, 50 Hz, pi/2));
  signal_source(in);

  source <electrical> wave_source("SOURCE2", cfg::across);
  wave_source(mains, in);
```

Self-oscillating half-bridge

Example

```
int sc_main (int argc, char *argv[])
{
  sc_core::sc_signal <bool> compout, pulse1, pulse2;
  sc_core::sc_signal <electrical::wave_type> in;

  ab_signal <electrical, parallel> mains, rectified;
  ab_signal <electrical, series> shunt;
  ab_signal <electrical, half_bridge> halfbridge;

  generator <electrical::wave_type> signal_source("SOURCE1",
                                  sine(sqrt(2) * 230 V, 50 Hz, pi/2));
  signal_source(in);

  source <electrical> wave_source("SOURCE2", cfg::across);
  wave_source(mains, in);
```

Self-oscillating half-bridge

Example

```
int sc_main (int argc, char *argv[])
{
  sc_core::sc_signal <bool> compout, pulse1, pulse2;
  sc_core::sc_signal <electrical::wave_type> in;

  ab_signal <electrical, parallel> mains, rectified;
  ab_signal <electrical, series> shunt;
  ab_signal <electrical, half_bridge> halfbridge;

  generator <electrical::wave_type> signal_source("SOURCE1",
                                  sine(sqrt(2) * 230 V, 50 Hz, pi/2));
  signal_source(in);

  source <electrical> wave_source("SOURCE2", cfg::across);
  wave_source(mains, in);
```

Self-oscillating half-bridge

Example

```
ideal_rectifier <electrical> bridge("BRIDGE");
bridge(mains, rectified);

RCs_load line_filter("FILTER", 1 ohm, 5 uF);
line_filter(rectified);

ab_connector <electrical> vdd("VDD");
vdd(halfbridge->mains, rectified);

onoff_switchd switch1("SWITCH1");
switch1(-halfbridge->up,pulse1);

onoff_switchd switch2("SWITCH2");
switch2(-halfbridge->down,compout);
```

Self-oscillating half-bridge

Example

```
ideal_rectifier <electrical> bridge("BRIDGE");
bridge(mains, rectified);

RCs_load line_filter("FILTER", 1 ohm, 5 uF);
line_filter(rectified);

ab_connector <electrical> vdd("VDD");
vdd(halfbridge->mains, rectified);

onoff_switchd switch1("SWITCH1");
switch1(-halfbridge->up,pulse1);

onoff_switchd switch2("SWITCH2");
switch2(-halfbridge->down,compout);
```

Self-oscillating half-bridge

Example

```
ideal_rectifier <electrical> bridge("BRIDGE");
bridge(mains, rectified);

RCs_load line_filter("FILTER", 1 ohm, 5 uF);
line_filter(rectified);

ab_connector <electrical> vdd("VDD");
vdd(halfbridge->mains, rectified);

onoff_switchd switch1("SWITCH1");
switch1(-halfbridge->up,pulse1);

onoff_switchd switch2("SWITCH2");
switch2(-halfbridge->down,compout);
```

Self-oscillating half-bridge

Example

```
ideal_rectifier <electrical> bridge("BRIDGE");
bridge(mains, rectified);

RCs_load line_filter("FILTER", 1 ohm, 5 uF);
line_filter(rectified);

ab_connector <electrical> vdd("VDD");
vdd(halfbridge->mains, rectified);

onoff_switchd switch1("SWITCH1");
switch1(-halfbridge->up,pulse1);

onoff_switchd switch2("SWITCH2");
switch2(-halfbridge->down,compout);
```

Self-oscillating half-bridge

Example

```
ideal_rectifier <electrical> bridge("BRIDGE");
bridge(mains, rectified);

RCs_load line_filter("FILTER", 1 ohm, 5 uF);
line_filter(rectified);

ab_connector <electrical> vdd("VDD");
vdd(halfbridge->mains, rectified);

onoff_switchd switch1("SWITCH1");
switch1(-halfbridge->up,pulse1);

onoff_switchd switch2("SWITCH2");
switch2(-halfbridge->down,compout);
```

Self-oscillating half-bridge

Example

```
  RLCs_2sm <electrical> load1("LOAD1", 4 ohm, 80 uH, 740 nF);
  load1(halfbridge->load, shunt);

  comparator <electrical, delayed> cmp("CMP",2e-8,cfg::through);
  cmp(shunt, compout);

  inverter inv1("INV1");
  inv1(compout,pulse1);

  sc_core::sc_start(sc_core::sc_time(400,sc_core::SC_US));
  return 0;
}
```

Self-oscillating half-bridge

Example

```
  RLCs_2sm <electrical> load1("LOAD1", 4 ohm, 80 uH, 740 nF);
  load1(halfbridge->load, shunt);

  comparator <electrical, delayed> cmp("CMP",2e-8,cfg::through);
  cmp(shunt, compout);

  inverter inv1("INV1");
  inv1(compout,pulse1);

  sc_core::sc_start(sc_core::sc_time(400,sc_core::SC_US));
  return 0;
}
```

Self-oscillating half-bridge

Example

```
  RLCs_2sm <electrical> load1("LOAD1", 4 ohm, 80 uH, 740 nF);
  load1(halfbridge->load, shunt);

  comparator <electrical, delayed> cmp("CMP",2e-8,cfg::through);
  cmp(shunt, compout);

  inverter inv1("INV1");
  inv1(compout,pulse1);

  sc_core::sc_start(sc_core::sc_time(400,sc_core::SC_US));
  return 0;
}
```

Self-oscillating half-bridge

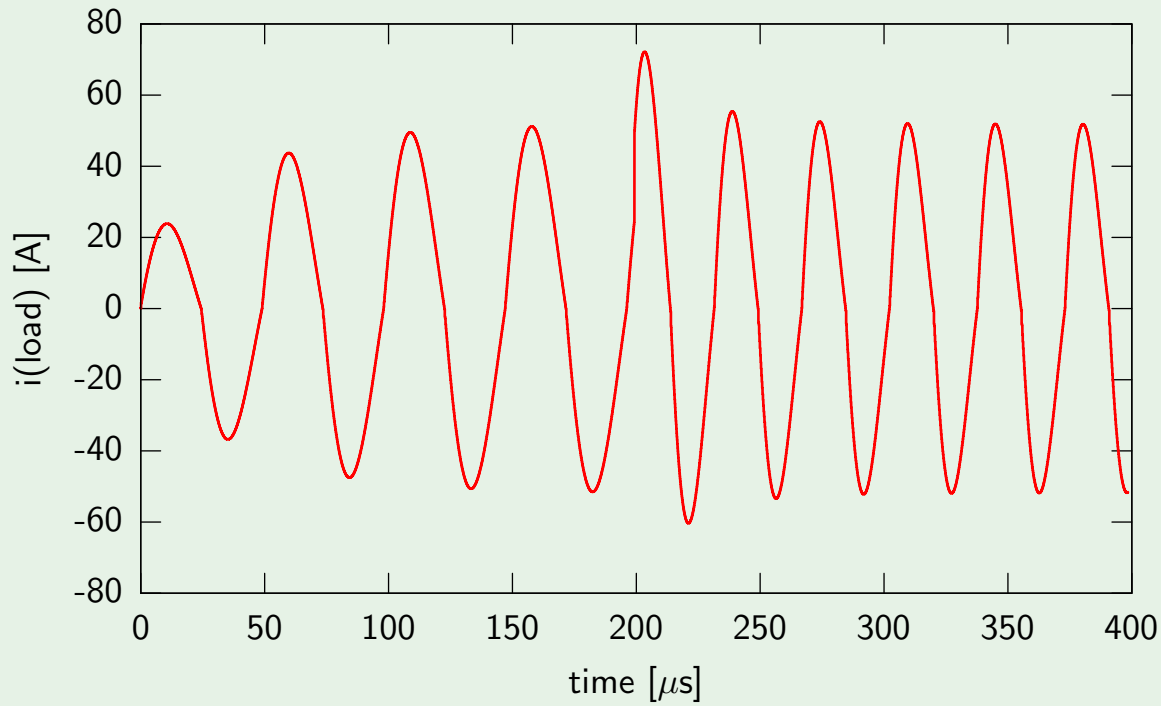
Example

```
  RLCs_2sm <electrical> load1("LOAD1", 4 ohm, 80 uH, 740 nF);
  load1(halfbridge->load, shunt);

  comparator <electrical, delayed> cmp("CMP",2e-8,cfg::through);
  cmp(shunt, compout);

  inverter inv1("INV1");
  inv1(compout,pulse1);

  sc_core::sc_start(sc_core::sc_time(400,sc_core::SC_US));
  return 0;
}
```

Outline

Simulation results

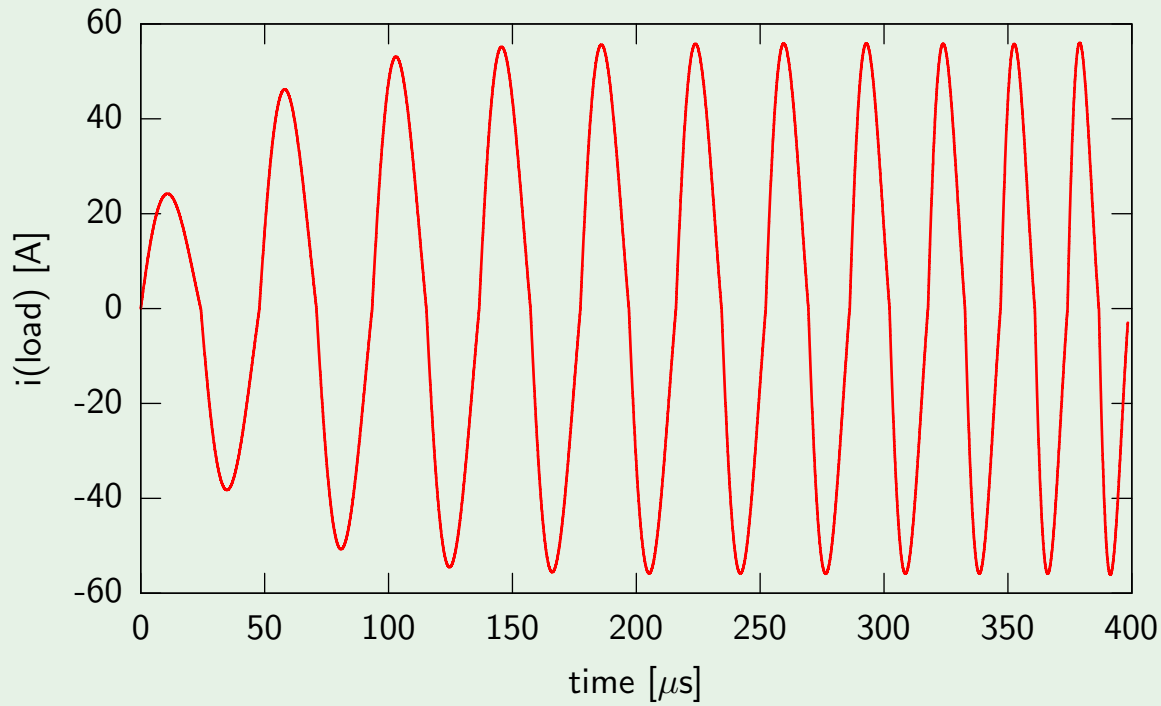
Current into the RLC load during power-on



After a 200 μs, an abruput change from 80 μH to 40 μH of the inductance of the load has been imposed.

Simulation results

Current into the RLC load during power-on

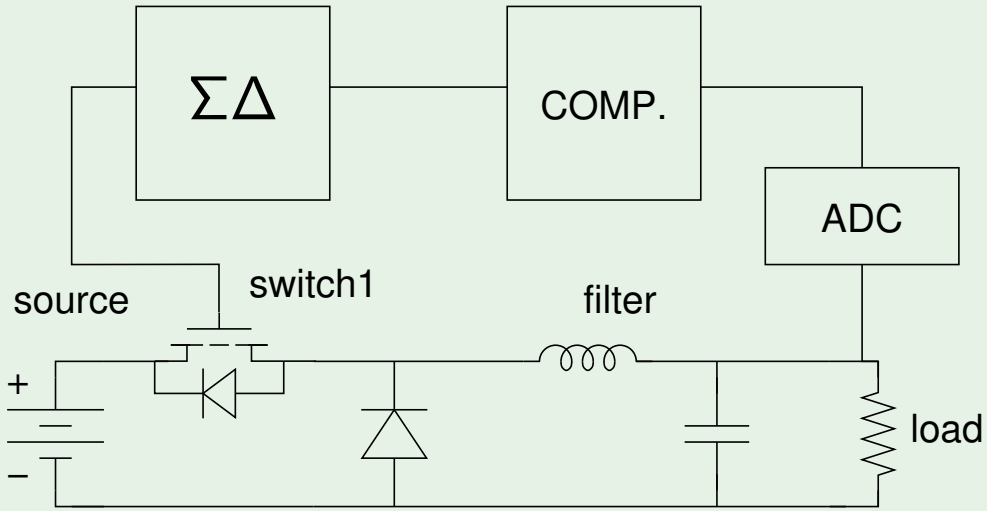


A sweep of the inductance of the load has been imposed changing continuosly the oscillating frequency.

Outline

Full Digital controlled DC-DC buck-converter

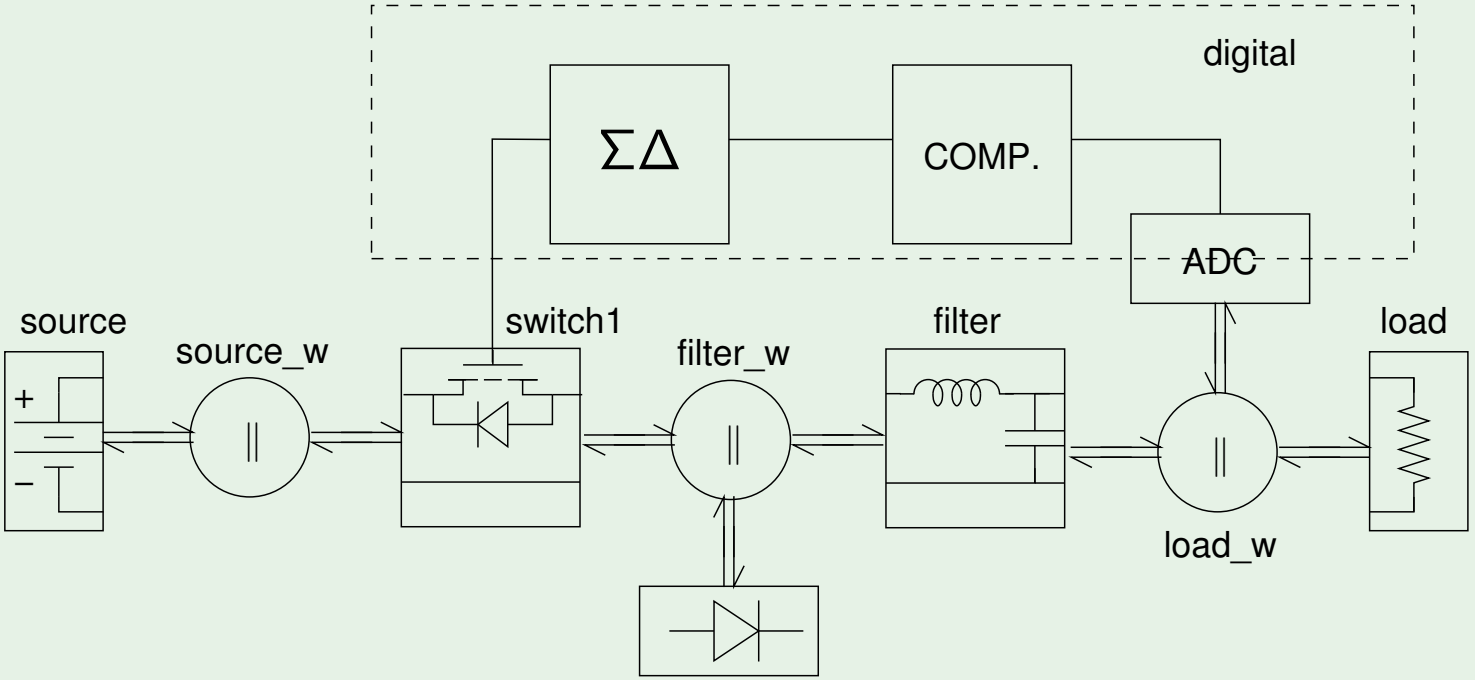
Example



Electrical schematic diagram

Full Digital controlled DC-DC buck-converter

Example



Buck-converter: SystemC-WMS modules and wavechannels interconnection diagram

Full Digital controlled DC-DC buck-converter

Example

```
int sc_main (int argc, char *argv[])
{
  sc_core::sc_signal <electrical::wave_type> in;
  sc_core::sc_signal <sc_fixed_fast<13,4> > s1;
  sc_core::sc_signal <sc_int<13> > s2;
  sc_core::sc_signal <bool> contr1;

  const sc_time t_PERIOD1 (5, SC_US);
  const sc_time t_PERIOD2 (1.25, SC_US);
  sc_core::sc_clock clk1("clk1", t_PERIOD1);
  sc_core::sc_clock clk2("clk2", t_PERIOD2);

  sc_core::sc_trace_file *f = create_tab_trace_file("LOAD");
  sc_core::sc_trace_file *f2 = create_tab_trace_file("LINE1");
  out.trace(f, "OUT");
  line1.trace(f2, "LINE1");
```

Full Digital controlled DC-DC buck-converter

Example

```
ab_signal <electrical, parallel> mains, line1, out;
generator <electrical::wave_type>
signal_source("SOURCE1", dc (14.4 V));
signal_source(in);

source <electrical> wave_source("SOURCE2", cfg::across);
wave_source(mains, in);

onoff_switchd_2s switch1("SWITCH1");
switch1(line1,mains, contr1);

diode diode1("DIODE1");
diode1(-line1);

LPF_II filter("FILTER", 50 uF, 50 uH);
filter(line1,out);
```

Full Digital controlled DC-DC buck-converter

Example

```
ab_signal <electrical, parallel> mains, line1, out;
generator <electrical::wave_type>
signal_source("SOURCE1", dc (14.4 V));
signal_source(in);

source <electrical> wave_source("SOURCE2", cfg::across);
wave_source(mains, in);

onoff_switchd_2s switch1("SWITCH1");
switch1(line1,mains, contr1);

diode diode1("DIODE1");
diode1(-line1);

LPF_II filter("FILTER", 50 uF, 50 uH);
filter(line1,out);
```

Full Digital controlled DC-DC buck-converter

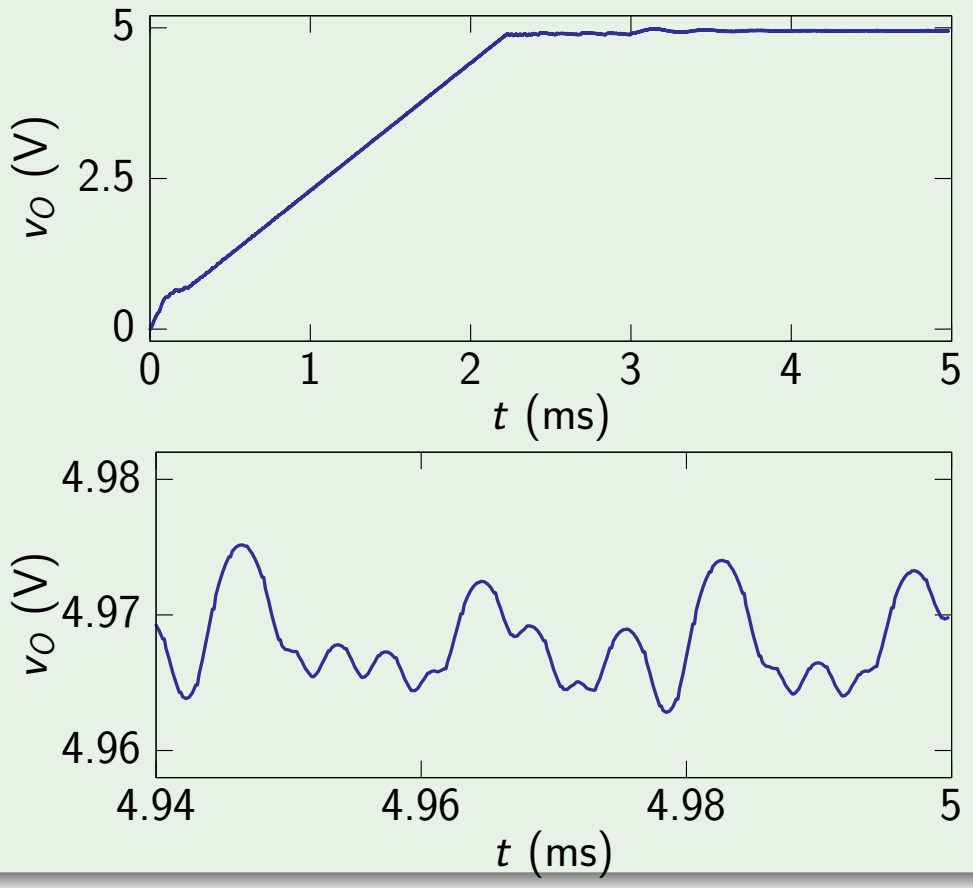
Example

```
R_load load("LOAD", 4.46 ohm);
load(out);

ADc adc1("ADC1");
adc1(out, s1, clk1);

compens comp1("COMP1");
comp1(s1, s2, clk1);

sigmadelta sd("SD");
sd(s2, clk2, contr1);

float sim_end;
sscanf(argv[1],"%f",&sim_end);
sc_core::sc_start(sim_end, sc_core::SC_SEC);
close_tab_trace_file(f);
close_tab_trace_file(f2);

return 0;
```

Outline

Simulation results

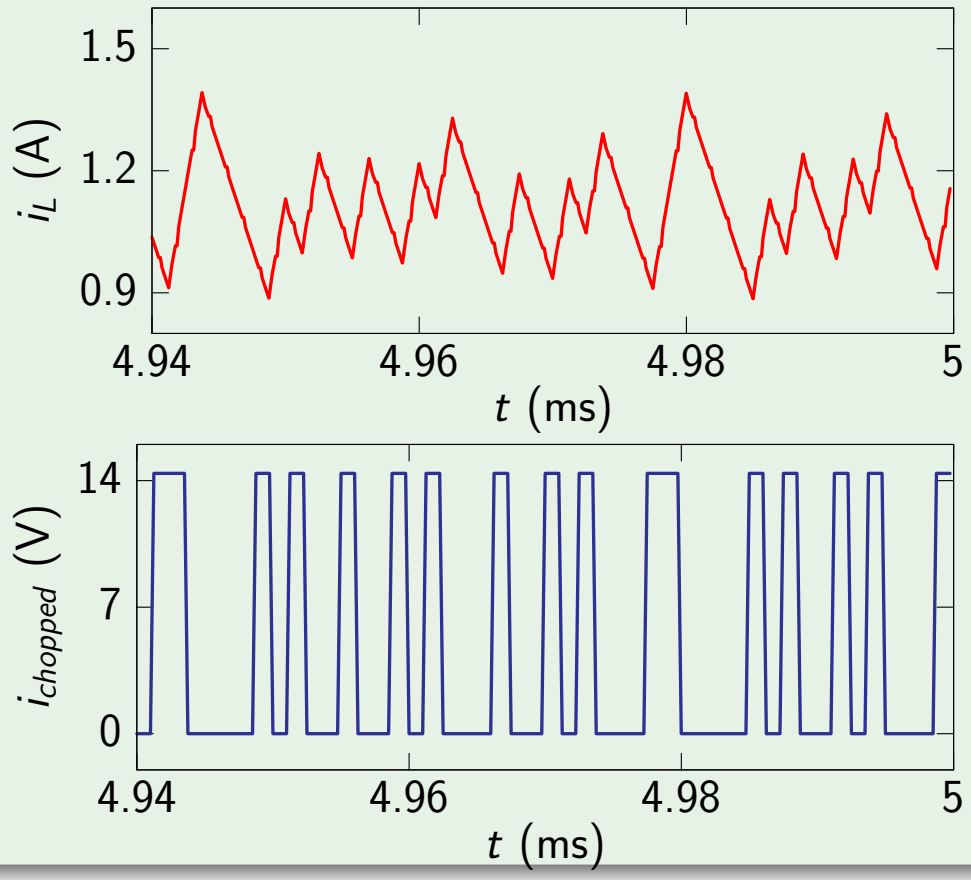
Output voltage



The load voltage during power-on and the load voltage in steady-state,

Simulation results

Current into inductor



The current in the inductor and the chopped voltage.

Outline

References and Conclusions

References

Giorgio Biagetti, Marco Caldari, Massimo Conti, and Simone Orcioni.

Extending SystemC to analog modeling and simulation.

In Christoph Grimm, editor, *Languages for system specification*, CHDL, chapter 15, pages 229–242. Kluwer Academic Publishers, 2004.

Simone Orcioni, Giorgio Biagetti, and Massimo Conti.

Systemc-WMS: Mixed signal simulation based on wave exchanges.

In Alain Vachoux, editor, *Applications of Specification and Design Languages for SoCs*, chapter 3, pages 43–59. Springer, 2006.

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/

Conclusions

SystemC-WMS main features:

- High-level simulation and description of mixed-signal systems
- Standard analog interface based on power exchange and designed for model reusability
- Seamless integration of complex heterogeneous systems
- Both linear and non-linear analog modelling capabilities
- Completely build using standard SystemC kernel features

For downloading the source code

- SourceForge
  http://www.sf.net
- SystemC-WMS home page.
  http://www.deit.univpm.it/systemc-wms/