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
1 Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2 Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
State of the Art

Simulation of heterogeneous systems comprising analog parts:

<table>
<thead>
<tr>
<th>SystemC-AMS</th>
<th>(Analog &amp; Mixed Signal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>· Modeling with block diagrams and directed signals</td>
<td></td>
</tr>
<tr>
<td>· Only linear systems can be modeled up to now</td>
<td></td>
</tr>
<tr>
<td>· Synchronous Data Flow MoC</td>
<td></td>
</tr>
<tr>
<td>· It is an OSCI effort</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SystemC-WMS</th>
<th>(Wave Mixed Signal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>· Wave theory-based “analog interfaces” used to interconnect blocks and automatically account for load effects</td>
<td></td>
</tr>
<tr>
<td>· Any locally-ODE nonlinear system can be modeled</td>
<td></td>
</tr>
<tr>
<td>· Automatic variable-time step ODE solver</td>
<td></td>
</tr>
<tr>
<td>· Fully compatible with SystemC-2.2.</td>
<td></td>
</tr>
</tbody>
</table>
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

1. Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
     - Wavechannel
     - SystemC-WMS Class Library

2. Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
High Level Modelling of Analog Modules

Modelling with nonlinear state space equations

\[
\begin{aligned}
    \dot{x} &= f(x, u) \\
y &= g(x, u)
\end{aligned}
\]

- x state
- u input
- y output
High Level Modelling of Analog Modules

**Modelling with nonlinear state space equations**

\[
\begin{align*}
\dot{x} &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

- **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
Modelling with nonlinear state space equations

\[
\begin{align*}
\dot{x} &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

- \(x\) 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
Modelling with nonlinear state space equations

\[
\begin{align*}
\dot{x} &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

- \(x\) 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
Modelling with nonlinear state space equations

\[
\begin{align*}
\dot{x} &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

- \( x \) 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

\[ \begin{align*}
\dot{x} &= f(x, u) \\
y &= g(x, u)
\end{align*} \]

- \( x \) state
- \( u \) input
- \( y \) output

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

\[ \begin{align*}
\dot{x} &= f(x, v) \\
i &= g(x, v)
\end{align*} \]
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
Two different modules that can be cascaded

\[ v_1 \quad i_1 \quad v_2 \quad i_2 \quad v_1 \quad i_1 \quad v_2 \quad i_2 \]

- 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

\[ v_1 \quad i_1 \quad v_2 \quad i_2 \quad v_1 \quad i_1 \quad v_2 \quad i_2 \]

- Some complex adaptor is needed to tie together the output voltages
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
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
Definition (Incident $a_j$ and Reflected $b_j$ wave)

\[
\begin{align*}
a_j &= \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} + i_j \sqrt{R_j} \right) \\
b_j &= \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} - i_j \sqrt{R_j} \right)
\end{align*}
\]

- $a_j^2 - b_j^2$ 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.
Definition (Incident $a_j$ and Reflected $b_j$ wave)

$$a_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} + i_j \sqrt{R_j} \right)$$

$$b_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} - i_j \sqrt{R_j} \right)$$

- $a_j^2 - b_j^2$ 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
**Definition (Incident $a_j$ and Reflected $b_j$ wave)**

\[ a_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} + i_j \sqrt{R_j} \right) \]
\[ b_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} - i_j \sqrt{R_j} \right) \]

- $a_j^2 - b_j^2$ 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.
**Definition (Incident \( a_j \) and Reflected \( b_j \) wave)**

\[
a_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} + i_j \sqrt{R_j} \right) \\
b_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} - i_j \sqrt{R_j} \right)
\]

- \( a_j^2 - b_j^2 \) 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)

$$a_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} + i_j \sqrt{R_j} \right)$$

$$b_j = \frac{1}{2} \left( \frac{v_j}{\sqrt{R_j}} - i_j \sqrt{R_j} \right)$$

- $a_j^2 - b_j^2$ 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

- $i_j$ and $v_j$ are input and output current and voltage, respectively.
- $a_j$ and $b_j$ are input and output waves, respectively.
- $a$ 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

\[ j \]

\[ i_j \]

\[ v_j \]

\[ j \]

\[ a_j \]

\[ b_j \]

- 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

1 Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2 Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
Wavechannel

Electrical schematic diagram

- analog module
- wave adaptor
- transmission line
- analog module
A wavechannel can be represented by a junction box connected to ports by transmission lines.
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
Fixed-point solution of DAE ⇔ Solution of Maxwell’s equations in quasi-static conditions ⇔ Kirchhoff’s equations
Consider a junction between $N$ ports

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

\[
\begin{cases} 
\mathbf{A}_v \mathbf{v} = 0 \\
\mathbf{A}_i \mathbf{i} = 0
\end{cases}
\]

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 $\pm 1$. Letting:

\[
\mathbf{A}_x = \mathbf{A}_v \text{ diag } R_k \quad \text{ and } \quad \mathbf{A}_y = \mathbf{A}_i \text{ diag } 1/R_k
\]

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

\[
\mathbf{S} = \begin{bmatrix} \mathbf{A}_x & -\mathbf{A}_x \\
\mathbf{A}_y & \mathbf{A}_y \end{bmatrix}^{-1}
\]
Parallel wavechannel

\[ \sum_{j=1}^{N} i_j = 0 \]

\[ v_1 = v_2 = \cdots = v_N \]

which leads to:

\[ A_v = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots \\ & \vdots & & & \vdots \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \]

\[ A_i = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \]

\[ N = 1 \implies \text{open circuit} \]
Series wavechannel

\[ \sum_{j=1}^{N} v_j = 0 \]

\[ i_1 = i_2 = \cdots = i_N \]

which leads to:

\[ A_v = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \]

\[ A_i = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots \\ \vdots \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \]

\[ N = 1 \ \Rightarrow \ \text{shunt} \]
Bridge — Half-bridge

\[ A_v = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 \\ 1 & 0 & 0 & 1 & -1 & 0 \end{bmatrix} \]

\[ A_i = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \]
Description and Modelling of Analog Modules in SystemC
- Modelling of Analog Modules Using Wave Quantities
- Wavechannel
- SystemC-WMS Class Library

Application examples
- Oscillator Schematics and SystemC Implementation
- Oscillator Simulation Results
- DC–DC Converter Schematics
- DC–DC Converter Simulation Results
- References and Conclusions
SystemC-WMS class library

a,b parameter modelling diagram
SystemC-WMS class library

SystemC-WMS class mapping

```
wave_module M1  ab_port
```

```
ab_signal
```

```
wave_module M2
```

```
C
```

```
ab_signal_if
```

```
M2  wave_module
```
SystemC-WMS class mapping
SystemC-WMS class library

SystemC-WMS class mapping

wave_module $M_1$ \( \leftrightarrow \text{ab_port} \)

ab_signal

wave_module $M_2$

wave_module

wave_module $M_2$

C

ab_signal_if
SystemC-WMS class library

SystemC-WMS class mapping

wave_module

M₁

ab_port

C

ab_signal

M₂

wave_module

wave_module

ab_signal_if

wave_module

M₂

M₂
SystemC-WMS class library

SystemC-WMS class mapping

wave_module \( M_1 \) \( \leftrightarrow \) ab_port \( \leftrightarrow \) wave_module

ab_signal

\( \downarrow \)

ab_signal_if

\( C \)

wave_module \( M_2 \) \( \leftrightarrow \) wave_module

wave_module \( M_2 \) \( \leftrightarrow \) wave_module
Example (\texttt{wave\_module})

```c
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*)

```c
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;
    }
};
```
Example (wave_module)

```c
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;
    }
};
```
Example (`wave_module`)

```cpp
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()) {
        double a = port->read(); // read incident wave here
        double b = g(x, a); // compute reflected wave
        port->write(b); // and send it out here
    }
}
```
Example (wave module)

```c++
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()) {
        double a = port->read(); // read incident wave here
        double b = g(x, a); // compute reflected wave
        port->write(b); // and send it out here
    }
}
```
Example \texttt{(wave\_module)}

```cpp
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()) {
        double a = port->read();  // read incident wave here
        double b = g(x, a);        // compute reflected wave
        port->write(b);            // and send it out here
    }
}
```
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()) {
        double a = port->read(); // read incident wave here
        double b = g(x, a);       // compute reflected wave
        port->write(b);       // and send it out here
    }
}
Example (wave\_module)

```c
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()) {
        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)

```cpp
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()) {
        double a = port->read(); // read incident wave here
        double b = g(x, a); // compute reflected wave
        port->write(b); // and send it out here
    }
}
```
**Example (wave_module)**

```cpp
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()) {
        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)

```c
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

1. Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2. Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
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

```c
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

```c
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);
}
```
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);
Example

```cpp
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**

```plaintext
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

```plaintext
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

```cpp
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

```plaintext
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

```systemc
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

```cpp
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");
in1(compout, pulse1);

sc_core::sc_start(sc_core::sc_time(400, sc_core::SC_US));
return 0;
```
Self-oscillating half-bridge

Example

```cpp
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

```cpp
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;
```
1. Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2. Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
After a 200 $\mu$s, an abrupt change from 80 $\mu$H to 40 $\mu$H of the inductance of the load has been imposed.
A sweep of the inductance of the load has been imposed changing continuously the oscillating frequency.
Outline

1. Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2. Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
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

```c
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");
}
```
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_IId 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);
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

1 Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2 Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
The load voltage during power-on and the load voltage in steady-state,
The current in the inductor and the chopped voltage.
Outline

1 Description and Modelling of Analog Modules in SystemC
   - Modelling of Analog Modules Using Wave Quantities
   - Wavechannel
   - SystemC-WMS Class Library

2 Application examples
   - Oscillator Schematics and SystemC Implementation
   - Oscillator Simulation Results
   - DC–DC Converter Schematics
   - DC–DC Converter Simulation Results
   - References and Conclusions
References

Giorgio Biagetti, Marco Caldari, Massimo Conti, and Simone Orcioni.
Extending SystemC to analog modeling and simulation.

Simone Orcioni, Giorgio Biagetti, and Massimo Conti.
Systemc-WMS: Mixed signal simulation based on wave exchanges.
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/
## 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 built 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/