TU Vienna SystemC AMS Communications Library

- Documentation -

Jiong Ou, Peter Brunmayr, Farooq Muhammad,
Jan Haase, Christoph Grimm
ou|brunmayr|farooq|haase|grimm@ict.tuwien.ac.at
Institute of Computer Technology
Vienna University of Technology

Contents

1 Overview

Simulation of signal processing systems and communication systems usually tends to be slow in standard simulation environments as the high frequencies lead to very small time steps and therefore many calculations. SystemC AMS offers modelling in Timed Data Flow (TDF) which allows faster simulations as the scheduling is done in advance. However, the modelling is a slow process as the system designer has to create most parts of his system “by hand”.

The TUV_AMS_LIBRARY is developed that provides building blocks to ease the design process, relieving the designer from their detailed implementation and giving him more time for the harder problems in his design. The implemented building blocks focus on the field of communication and radio frequency systems, in particular on signal sources, modulation/demodulation blocks, filters, measurement and observation parts. They can be subdivided into three categories of blocks:

- Signal sources
- Basic blocks for signal processing
- Signal analysis units

Currently the library components are augmented with means for specification of re-configurability, and means to model physical implementations of re-configurability where possible.

In the next Sections all of the implemented modules will be described uniformly with:

- Functional description
- Module definition
- Module interface
- Adjustable parameters
- Module implementation if necessary
- Simulation example with results by several modules

After the description of all of the implemented modules several relative complex communication system are built up and simulated using modules from the TUV_AMS_LIBRARY to present the usage of the library and how it can improve the efficiency of designing and modelling of communication systems.

2 Provided Models

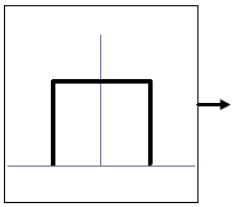
In this section the implemented building blocks are described in detail. They consist of parameterizable SystemC AMS modules.

2.1 Signal Sources

This section describes several classes of signal sources.

2.1.1 Uniformly distributed random Numbers

This class generates uniformly distributed random numbers. The parameters _min and _max are used to set the upper and lower bound of the output values. The data rate can be increased by the parameter _rate.

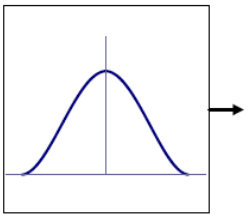


Class definition: noise_uniform(sc_core::sc_module_name nm, double _min, double _max,
int _rate);
Interfaces: sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name		
_min	double	1.0	smallest output value
_max	double	0.0	biggest output value
_rate	integer	1	data rate of output port

2.1.2 Gaussian distributed random Numbers

This class generates Gaussian distributed random numbers. The parameters _mean and _variance are used to set the mean and variance of the random numbers. The data rate can be increased with the parameter _rate.

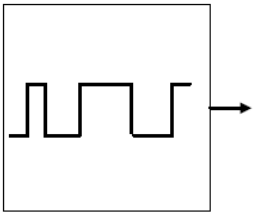


Class definition: noise_gauss(sc_core::sc_module_name nm, double _variance, double
_mean, int _rate);
Interfaces: sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name		
_variance	double	1.0	variance
_mean	double	0.0	mean
_rate	integer	1	data rate of input and output port

2.1.3 Uniformly distributed random Bits

This class generates a uniformly distributed random sequence of bits on its output.

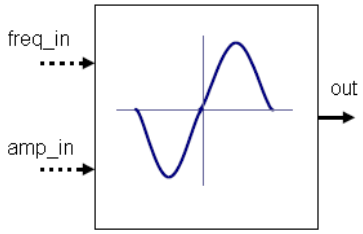


Class definition: rand_bool(sc_core::sc_module_name nm, int _rate);
Interfaces: sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name		
_rate	integer	1	data rate output port

2.1.4 Sine

The output of this class is a sine wave. The frequency, amplitude, offset and the initial phase of the output can be set with accordant parameters. You can also change the frequency and/or amplitude of the sine wave during simulation using “freq_in” and “amp_in” ports. In this case parameter “freq_c” (frequency configuration) and/or “amp_c” (amplitude configuration) should be set to “true”. Otherwise “freq_in” and/or “amp_in” are/or not available. Meanwhile you must add the “&” symbol before name of signals which are connected to these two ports (see example).



Class definition: sine(sc_core::sc_module_name n, double freq_def, double amp_def, double
_phi, double _offset, bool amp_c, bool freq_c,int datarate);
Interfaces: sca_tdf::sca_out<double> out;
sca_tdf::sca_in<double> freq_con;
sca_tdf::sca_in<double> amp_con;

Parameter	Type	Default value	Description
nm	sc_module_name		
freq_def	double	1.0e3	initial frequency
amp_def	double	1.0	initial amplitude
_phi	double	0.0	initial phase
_offset	double	0.0	output offset
amp_c	bool	false	amplitude configurable
freq_c	bool	false	frequency configurable
datarate	integer	1	Increases data rate

Example :

```
 sine sin("sin",1.0e3 , 1.0 , 0.0 , 0.0 , true , true , 1);
sin.freq_con(&sig_freq);
sin.amp_con(&sig_amp);
sin.out(signal_out);
```

Note: By instantiating module "&" must be added to the signals, if the signals are connected to amp_in or freq_in ports.

2.1.5 Square Wave

The output of this class is a square wave. With the parameter "freq" and "amp" user can define the frequency and amplitude of the output signal. The parameter "ofs" is used to set the offset of the output signal.

Class definition: sqr_gen(sc_core::sc_module_name n, double freq, double amp, double ofs, int rate);
Interface: sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name		
freq	double	-	frequency of the generated wave
amp	double	-	amplitude of the generated wave
ofs	double	0.0	output offset
rate	integer	1	output data rate

2.1.6 Triangle Wave

The output of this class is a triangle wave. With the parameter "freq" and "amp" user can define the frequency and amplitude of the output signal. The parameter "ofs" is used to set the offset of the output signal.

Class definition: tri_gen(sc_core::sc_module_name n, double freq, double amp, double ofs,
int rate);
Interface: sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name		
freq	double	-	frequency of the generated wave
amp	double	-	amplitude of the generated wave
ofs	double	0.0	output offset
rate	integer	1	output data rate

2.1.7 Saw tooth Wave

The output of this class is a saw tooth wave. With the parameter “freq” and “amp” user can define the frequency and amplitude of the output signal. The parameter “ofs” is used to set the offset of the output signal.

Class definition: saw_gen(sc_core::sc_module_name n, double freq, double amp, double
ofs, int rate);
Interface: sca_tdf::sca_out<double> out;

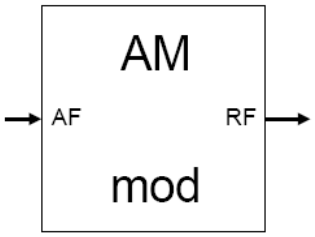
Parameter	Type	Default value	Description
nm	sc_module_name		
freq	double	-	frequency of the generated wave
amp	double	-	amplitude of the generated wave
ofs	double	0.0	output offset
rate	integer	1	output data rate

2.2 Modulation/Demodulation

This section describes classes for modulation and demodulation of signals.

2.2.1 Amplitude Modulation

This class modulates the input signal to a carrier. The carrier frequency, amplitude and initial phase can be set with parameters. The parameter _offset adds a constant value to the input signal. It is possible to increase the output data rate with the _rate parameter.

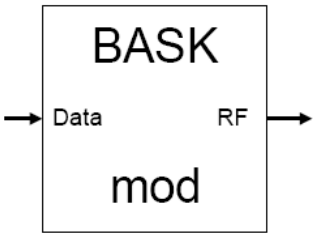


Class definition: mod_am(sc_core::sc_module_name nm, double _freq, double _ampl,
double _offset, int _rate) ;
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	-	carrier frequency
_ampl	double	1.0	carrier amplitude
_offset	double	0.0	output offset
_rate	integer	1	increases data rate

2.2.2 Binary Amplitude Shift Keying (BASK) Modulatorfre

This class modulates an input bit stream to a carrier using binary amplitude shift keying. Frequency, amplitude and initial phase of the carrier can be set with parameters. Parameter "_ampl1" and "_ampl0" define the carrier amplitude for binary value 1 and 0, respectively.



Class definition: mod_bask(sc_core::sc_module_name nm, double _freq, double _ampl1,
double _ampl0, double _phi, int _rate) ;
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	-	carrier frequency
_ampl1	double	1.0	carrier amplitude transmit 1
_ampl0	double	0.0	carrier amplitude transmit 0
_phi	double	0.0	initial phase
_rate	integer	1	increases data rate

2.2.3 BASK Demodulator

This class demodulates a high frequency signal to a bit stream. Frequency of carrier signal can be set with parameter "_freq". A threshold signal level has to be set in order to let the module works correctly. If detected signal is larger than the threshold value the module will output "true" and vice verse.

Class definition: demod_bask(sc_core::sc_module_name nm, double _levle,
double _freq = 10.0e3, int _rate = 1);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_level	double	-	threshold value
_freq	double	1000.0	carrier frequency
_rate	integer	1	increases data rate

2.2.4 Binary Phase Shift Keying (BPSK) Modulator

This class modulates an input bit stream to a carrier using binary phase shift keying. Frequency of the carrier and data rate of output port can be set with parameters.

Class definition: bpsk(sc_core::sc_module_name n, double _freq=1000,int _out_rate=1);
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
_out_rate	integer	1	increases data rate of output port

2.2.5 BPSK Demodulator

This class demodulates a BPSK modulated signal to a bit stream. Frequency of the carrier and data rate of input port can be set with parameters.

Class definition: bpsk_de(sc_core::sc_module_name n, double _freq=1000, int _in_rate=1);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
_in_rate	integer	1	increases data rate of input port

2.2.6 DBPSK Modulator

This class modulates an input bit stream to a carrier using differential binary phase shift keying. Frequency of the carrier and data rate of output port can be set with parameters.

Class definition: dbpsk_de(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out**;**

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of output port

2.2.7 DBPSK Demodulator

This class demodulates a DBPSK modulated signal to a bit stream. Frequency of the carrier and data rate of input port can be set with parameters.

Class definition: dbpsk_de(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of input port

2.2.8 QPSK Modulator

This class modulates an input bit stream to a carrier using quadrature phase shift keying. Frequency of the carrier and data rate of output port can be set with parameters.

Class definition: qpsk(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of output port

2.2.9 QPSK Demodulator

This class demodulates a QPSK modulated signal to a bit stream. Frequency of the carrier and data rate of input port can be set with parameters.

Class definition: qpsk_de(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of input port

2.2.10 OQPSK Modulator

This class modulates an input bit stream to a carrier using offset quadrature phase shift keying. Frequency of the carrier and data rate of output port can be set with parameters.

Class definition: oqpsk(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of output port

2.2.11 OQPSK Demodulator

This class demodulates a OQPSK modulated signal to a bit stream. Frequency of the carrier and data rate of input port can be set with parameters.

Class definition: oqpsk_de(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of input port

2.2.12 DQPSK Modulator

This class modulates an input bit stream to a carrier using differential quadrature phase shift keying. Frequency of the carrier and data rate of output port can be set with parameters.

Class definition: dqpsk(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of output port

2.2.13 DQPSK Demodulator

This class demodulates a DQPSK modulated signal to a bit stream. Frequency of the carrier and data rate of input port can be set with parameters.

Class definition: dqpsk_de(sc_core::sc_module_name n, double _freq, int rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	1000.0	carrier frequency
rate	integer	1	increases data rate of input port

2.2.14 M-FSK Modulator

The multiple FSK modulator transmits signal through discrete frequency changes of a carrier wave. The lowest carrier frequency and the frequency interval between two adjacent carriers can be set per the parameter _freq_basis and _freq_interval, respectively. With the template N you can set the bit width of input port. The frequencies will be than assigned to each symbol automatically. For instance, by setting N=2, _freq_basis=1000 and _freq_interval=1000, the frequency 1000 will be assigned to symbol “00” , the frequency 2000 will be assigned to symbol “01” and so on. It is also possible to increase the output data rate with the _out_rate parameter.

Class definition: template <int N>
fsk(sc_core::sc_module_name n, double _freq_basis, double _freq_interval, int _out_rate)
Interfaces: sca_tdf::sca_in<sc_bv<N> > in; (Using template N to define the bitwidth of input port)
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
_freq_basis	double	1000	the lowest carrier frequency
_freq_interval	double	1000	frequency interval between two adjacent carriers
_out_rate	integer	1	data rate of output port

Example:
fsk<4> fsk("fsk",1000,500,1);
fsk.in(sig_stim);
fsk.out(sig_out);

In this example a fsk modulator with 4 bits wide input port is instanced. The lowest carrier frequency and frequency interval are set to 1000 and 500, respectively.

2.2.15 M-PSK Modulator

The multiple PSK modulator conveys data by changing, or modulating, the phase of a reference signal (the carrier wave). The carrier frequency can be set per the parameter _freq. With the template N you can set the bit width of input port. The phases will be than assigned to each symbol automatically.

Class definition: template <int N>
psk(sc_core::sc_module_name n, double _freq, int _out_rate=1)
Interfaces: sca_tdf::sca_in<sc_bv<N> > in;

sca_tdf::sca_out<double> out;

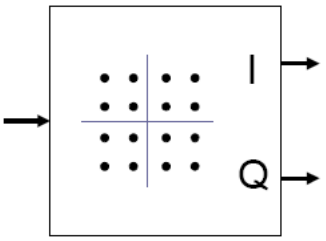
Parameter	Type	Default value	Description
n	sc_module_name	-	
_freq	double	-	carrier frequency
_out_rate	integer	1	data rate of output port

Example:
psk<4> psk("psk",1000,1);
fsk.in(sig_stim);
fsk.out(sig_out);

In this example a psk modulator with 4 bits wide input port is instanced. Phase "$i \cdot \frac{2\pi}{16}$" will be assigned to the ith symbol.

2.2.16 QAM Mapper

This class maps an input bitstream to a rectangular IQ constellation [1]. 4, 16, 64 and 256 point constellations are implemented. The data rate of the output is 2 (resp. 4, 6, 8) times smaller then the data rate of the input. At the moment it is not possible to define arbitrary QAM constellations. This will be implemented in the future.

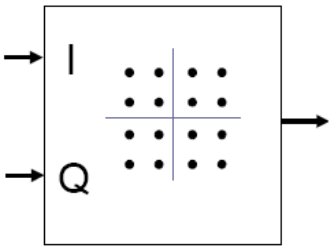


Class definition: qam_map(sc_core::sc_module_name nm, int _rate);
Interfaces: sca_tdf::sca_in<bool> in ;
sca_tdf::sca_out<double> out_i ;
sca_tdf::sca_out<double> out_q ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_rate	integer	4	Number of points in the constellation (4,16,64,256)

2.2.17 QAM Demapper

This class maps a pair of double value inputs, namely I and Q signal, per a rectangular IQ constellation to an output bitstream [1]. 4, 16, 64 and 256 point constellations are implemented. The data rate of the input is 2 (resp. 4, 6, 8) times smaller then the data rate of the output. At the moment it is not possible to define arbitrary QAM constellations. This will be implemented in the future.

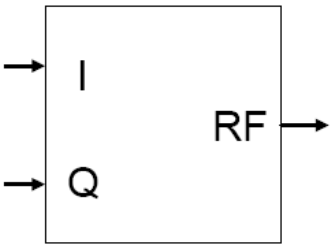


Class definition: qam_demap(sc_core::sc_module_name nm, int _rate);
Interfaces: sca_tdf::sca_in<double> in_i ;
sca_tdf::sca_in<double> in_q ;
sca_tdf::sca_in<bool> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_rate	integer	4	Number of points in the constellation (4,16,64,256)

2.2.18 IQ Modulator(will be removed)

This class modulates the IQ input signals to a carrier frequency. The optional parameters _d_ampl and _d_phi allows to simulate phase and amplitude errors of the IQ signals. It is possible to increase the output data rate with the _rate paramter.

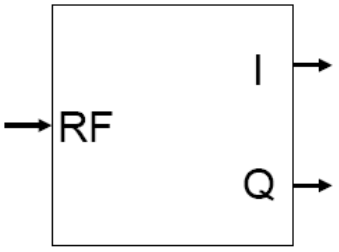


Class definition: mod_iq(sc_core::sc_module_name nm, double _freq, double _d_ampl, double _d_phi, int _rate);
Interfaces: sca_tdf::sca_in<double> i ;
sca_tdf::sca_in<double> q ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	-	output frequency
_d_ampl	double	0.0	amplitude error
_d_phi	double	0.0	phase error
_rate	integer	1	increases data rate

2.2.19 IQ Demodulator(will be removed)

This class demodulates a carrier and writes the demodulated baseband IQ signals to the outputs. The parameter _freq defines the frequency of the input carrier. IQ mismatches can be added with the parameters _d_ampl and _d_phi.

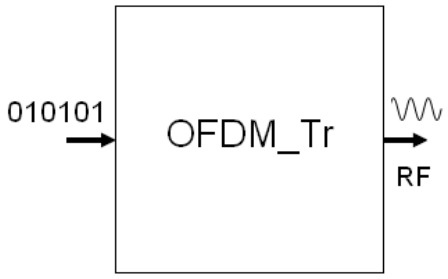


Class definition: demod_iq(sc_core::sc_module_name nm, double _freq, double _d_ampl, double _d_phi);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> i ;
sca_tdf::sca_out<double> q ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_freq	double	-	output frequency
_d_ampl	double	0.0	amplitude error
_d_phi	double	0.0	phase error
_rate	integer	1	decreases data rate

2.2.20 OFDM modulator (OFDM Transmitter)*

This class modulates the digital input serial signal to an OFDM carrier signal, which is the sum of a number of orthogonal sub-carriers, with baseband data on each sub-carrier being independently modulated using quadrature amplitude modulation (QAM). The bit width of symbol can be set with the template N.



Class definition: template <int N>
ofdm_se(sc_core::sc_module_name n, double mixer_fc, int qam_p_num, double bit_f, int dout_rate, double _amp)
Interfaces: sca_tdf::sca_in<bool> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
mixer_fc	double	-	Carrier frequency
qam_p_num	int	-	Dimension of QAM-Constellation
bit_f	double	-	Bit rate of bit streams on the input port

dout_rate*	integer	-	Data rate of output port
_ampl	double	1.0	Amplitude of carrier wave

* **dout_rate:** Note that this datarate can basically be set independently. But a value too low would result in an undesirable outcome regarding Nyquist's sampling theorem. The parameter dout_rate ensures the number of tokens per sine period on the output port.

2.2.21 OFDM demodulator (OFDM Receiver)

This class demodulates an OFDM carrier signal to a digital signal. The bit width of symbol can be set with the template N.

Class definition: template <int N>
ofdm_re(sc_core::sc_module_name n, double mixer_fc, int demap_p, double bit_f, int dout_rate, double _amp)

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<bool> out;

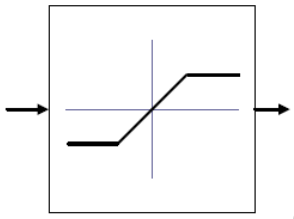
Parameter	Type	Default value	Description
nm	sc_module_name	-	
mixer_fc	double	-	Carrier frequency
deqam_p	int	-	Dimension of QAM-Demapper Constellation. It should be equal to qam_p_num of the transmitter
bit_f	double	-	Bit rate of the bit stream on the input port of the sender
dout_rate	integer	-	Data rate of input port. This value should be equal to the dout_rate of the transmitter.
_ampl	double	-	Amplitude of carrier wave

2.3 Nonlinearities

2.3.1 Saturation

This module "clips" a signal value if it exceeds certain extremal values, i.e, it implements the function:

$$f_{high,low}(x) = \begin{cases} x & if \quad low \le x \le high \\ high & if \quad high > x \\ low & if \quad low < x \end{cases}$$



(to be korrigiert)

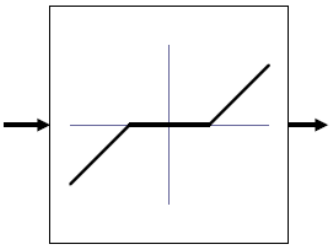
Class definition: saturation(sc_core::sc_module_name nm, double _high, double _low);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_high	double	5.0	minimum output value
_low	double	0.0	maximum output value

2.3.2 Deadzone

This module implements the following function:

$$f_{high,low}(x) = \begin{cases} 0 & if \quad low \le x \le high \\ x - high & if \quad x > high \\ x + low & if \quad x < low \end{cases}$$

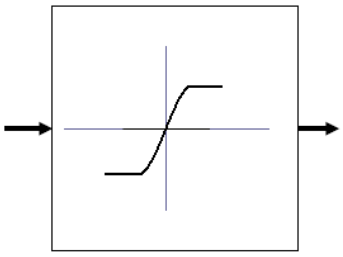


Class definition: deadzone(sc_core::sc_module_name nm, double _high, double _low);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_high	double	0.1	upper band limit
_low	double	-0.1	Lower band limit

2.3.3 Tan function

This module implements the following function:



$$f(x) = \tanh(x)$$

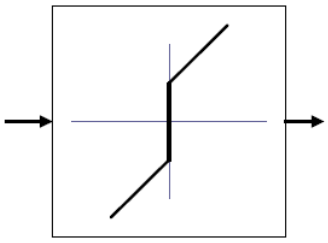
Class definition: tan(sc_core::sc_module_name n);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	

2.3.4 Coulomb function

This class implements the coulomb function which is defined as:

$$f_{gain,y}(x) = gain \cdot x + y \cdot sign(x)$$



Class definition: coulomb(sc_core::sc_module_name nm, double _gain, double _y);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

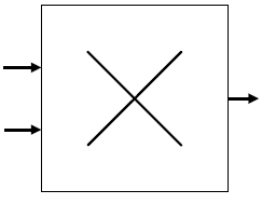
Parameter	Type	Default value	Description
nm	sc_module_name	-	
_gain	double	1.0	factor
_y	double	-0.1	point of the y-intercept

2.4 Math

This section describes modules that provide often used math functions.

2.4.1 Multiplier

This module multiplies the value of input in1 to the value of input in2 and writes the result to the output out.



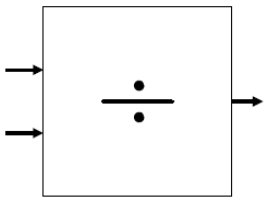
Class definition: mul(sc_core::sc_module_name nm);
Interfaces: sca_tdf::sca_in<double> in1 ;
sca_tdf::sca_in<double> in2 ;

sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	

2.4.2 Division

This module divides the value of input in1 from the value of input in2 and writes the result to the output out.

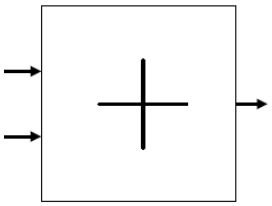


Class definition: divi(sc_core::sc_module_name nm);
Interfaces: sca_tdf::sca_in<double> in1 ;
sca_tdf::sca_in<double> in2 ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	

2.4.3 Adder

This module adds the value of input in1 to the value of input in2 and writes the result to the output out.

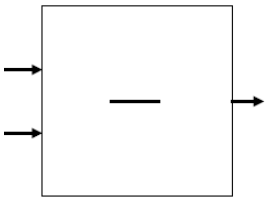


Class definition: add(sc_core::sc_module_name nm);
Interfaces: sca_tdf::sca_in<double> in1 ;
sca_tdf::sca_in<double> in2 ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	

2.4.4 Subtraction

This module subtracts the value of input in2 from the value of input in1 and writes the result to the output out.

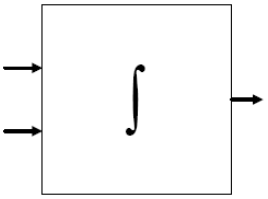


Class definition: sub(sc_core::sc_module_name nm);
Interfaces: sca_tdf::sca_in<double> in1 ;
sca_tdf::sca_in<double> in2 ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	

2.4.5 Integrator

This module integrates its input signal. An initial condition can be set with the parameter _initial.



Class definition: integ(sc_core::sc_module_name nm, double _initial);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_initial	double	0.0	initial condition

2.4.6 Logarithm

This module calculates the logarithm for an applied input signal. The base of the logarithm can be set with the parameter "_base".

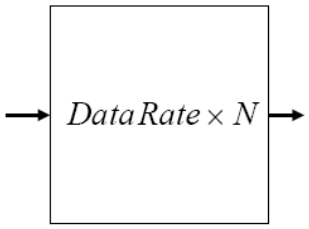
Class definition: logarithm(sc_core::sc_module_name n, string _base);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_base	string	"10"	base of the logarithm

2.5 Miscellaneous

2.5.1 Up-Sample

This module increases the data rate of the input signal. It reads one value and writes it _rate times to the output.

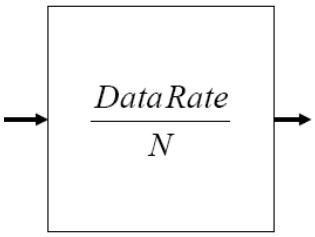


Class definition: upsample(sc_core::sc_module_name nm, int _rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_rate	integer	1	Increase data rate

2.5.2 Down-Sample

This module decreases the data rate of the input signal. It reads ***_rate*** values and writes one value to the output. With the parameter ***_sel*** it is possible to select the data which you want to output. It is an integer number between 1 and ***_rate***.



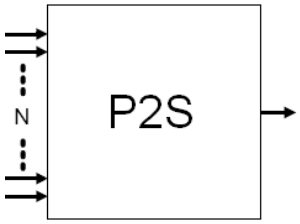
Class definition: downsample(sc_core::sc_module_name nm, int _sel, int _rate);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_sel	integer	1	the _sel th data of the input data
_rate	integer	1	decrease data rate

2.5.3 Parallel to Serial

This module converts n streams of parallel data to a serial data stream. The number of input ports can be set with the template N. The data type can be set with the

template T. The parameter ***_in_rate*** defines the number of data that will be read from one port at one time.



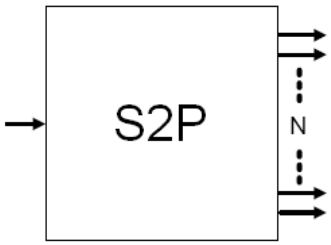
Class definition: template <class T, int N>
p2s(sc_core::sc_module_name n, int _in_rate);

Interfaces: sca_tdf::sca_in<T> in[N];
sca_tdf::sca_out<T> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_in_rate	integer	1	increase data rate

2.5.4 Serial to Parallel

This module converts a serial data stream to n streams of parallel data. The number of output ports can be set with the template N. The data type can be set with the template T. The parameter ***_out_rate*** defines the number of data that will be written from one port at one time.



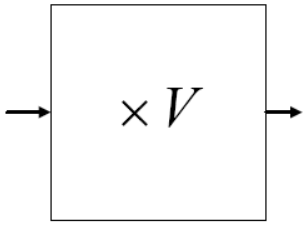
Class definition: template <class T, int N>
s2p(sc_core::sc_module_name n, int _out_rate);

Interfaces: sca_tdf::sca_in<T> in[N];
sca_tdf::sca_out<T> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_out_rate	integer	1	increase data rate

2.5.5 Gain

This module multiplies a constant factor (parameter _gain) to the input signal.

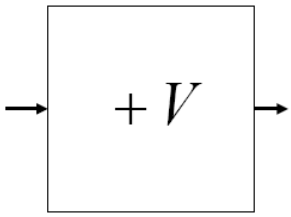


Class definition: gain(sc_core::sc_module_name nm, double _gain);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_gain	double	1.0	factor

2.5.6 Offset

This module adds a constant offset value (parameter _offset) to the input value and writes the result to the output.



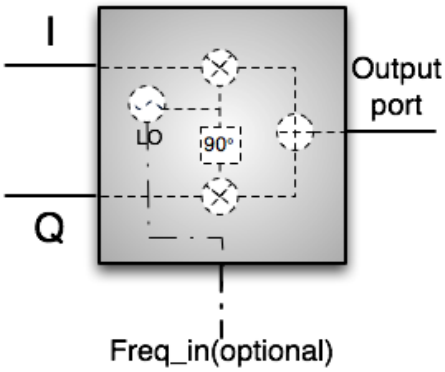
Class definition: offset(sc_core::sc_module_name nm, double _offset);
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<double> out ;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_offset	double	1.0	Offset value

2.6 Quadrature Mixer

2.6.1 Q_Mixer for transmitter

An up conversion system which can be applied easily for I/Q signal systems. You can also change the frequency of the local oscillator during simulation using “freq_in” port. In this case parameter “f_config” must be set to “true” by the instantiation of the module. Otherwise “freq_in” is not available.



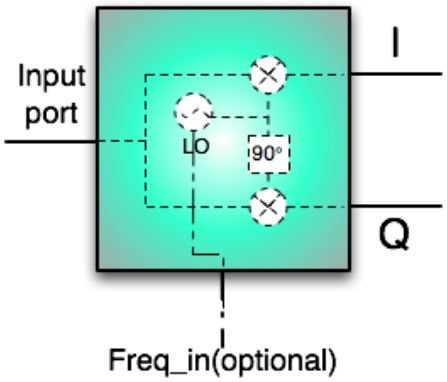
Class definition: q_mixer_re(sc_core::sc_module_name n,double _freq,int data_rate, bool f_config);

Interfaces: sca_tdf::sca_out<double> out; // output of mixer
sca_tdf::sca_in<double> i_in; // input of I signal
sca_tdf::sca_in<double> q_in; // input of Q signal
sca_tdf::sca_in<double>* freq_in; // optional input polrt of frequency

Parameter	Type	Default value	Description
n	sc_module_name	-	
_freq	double	-	central frequency(initial frequency of local oscillator)
data_rate	integer	1	data rate of input and output ports
f_config	bool	false	frequency configurable

2.6.2 Q_Mixer for receiver

A down conversion system which can be applied easily for I/Q signal systems. You can also change the frequency of the local oscillator during simulation using “freq_in” port. In this case parameter “f_config” must be set to “true” by the instantiation of the module. Otherwise “freq_in” is not available.



Class definition: q_mixer_re(sc_core::sc_module_name n,double _freq,int data_rate, bool f_config);

Interfaces: sca_tdf::sca_in<double> in; // input of mixer
sca_tdf::sca_out<double> i_out; // output of I signal
sca_tdf::sca_out<double> q_out; // output of Q signal
sca_tdf::sca_in<double>* freq_in; // optional input port of frequency

Parameter	Type	Default value	Description

n	sc_module_name	-	
_freq	double	-	central frequency(initial frequency of local oscillator)
data_rate	integer	1	data rate of input and output ports
f_config	bool	false	frequency configurable

2.7 Signal splitter

With this module user can easily copy the input signal to several output ports. The signal type (bool, double etc.) and the number of output ports can be set with the template T and N, respectively.

Class definition: template <class T, int N>
splitter(sc_core::sc_module_name n)

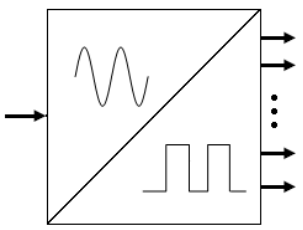
Interfaces: sca_tdf::sca_in<T> in;
sca_tdf::sca_out<T> out[N];

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_gain	double	1.0	Offset value

2.8 A/D and D/A converter

2.8.1 General n-bit A/D converter

This class converts continuous signals to N bit discrete digital numbers. The reference voltage can be set with parameter ***uref***. The optional parameters ***gain_e*** and ***offset_e*** allows to simulate the static errors like gain error and offset error. Their unit is LSB. The parameter ***gain_e defines*** the maximal gain error and respectively ***offset_e*** defines the maximal offset error. By the instantiation of the model a value between –gain_e LSB and +gain LSB (or –offset_e LSB and +offset_e LSB) will be selected automatically according to the uniform distribution. The number of the output ports (resolution of the converter) can be set with the template ***N***. Please note that the highest bit is used to express the algebra sign of the output value. So, one extra bit must be reserved for it. For instance, N should be set to 4 when a resolution of 3 bit is expected. (The INL and DNL error will be implemented in the future.)



Class definition: template <int N>
adc(sc_core::sc_module_name n, double uref, double gain_e, double offset_e);

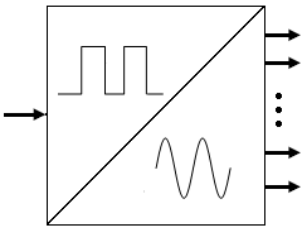
Interfaces: sca_tdf::sca_in<double> in ;
sca_tdf::sca_out<sc_bv<N> > out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
uref	double	1.0	reference voltage of ADC
gain_e	double	0.0	maximum gain error
offset_e	double	0.0	maximum offset error

Example:
adc<4> i_adc ("adc",2.0,0.5,0.7);
i_adc.out(signal_out);
i_adc.in(signal_in);

2.8.2 General n-bit D/A converter

This class converts a digital code to an analog signal. The reference voltage can be set with parameter ***uref***. The optional parameters ***gain_e*** and ***offset_e*** allows to simulate the static errors like gain error and offset error. Their unit is LSB. The parameter ***gain_e defines*** the maximal gain error and respectively ***offset_e*** defines the maximal offset error. By the instantiation of the model a value between –gain_e LSB and +gain LSB (or –offset_e LSB and +offset_e LSB) will be selected automatically according to the uniform distribution. The number of the output ports (resolution of the converter) can be set with the template ***N***. Please note that the highest bit is used to express the algebra sign of the output value. So, one extra bit must be reserved for it. For instance, N should be set to 4 when a resolution of 3 bit is expected. (The INL and DNL error will be implemented in the future.)



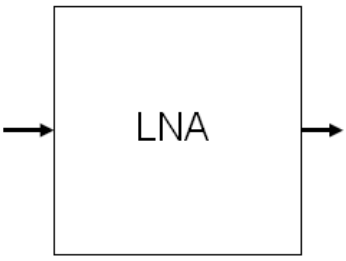
Class definition: dac(sc_core::sc_module_name n, double uref, double gain_e, double offset_e)
Interfaces: sca_tdf::sca_in<sc_bv<N> > in;
sca_tdf::sca_in<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
uref	double	1.0	reference voltage of DAC
gain_e	double	0.0	maximum gain error
offset_e	double	0.0	maximum offset error

Example:
dac<4> i_dac ("dac",2.,0.,0.);
i_dac.in(dig_out);
i_dac.out(ana_out);

2.9 LNA (Low-noise amplifier)

This class amplifies an input signal with a certain gain. User can set the gain with the parameter _gain. This class also allows user to model the intercept modulation of LNA. With the parameter "_ideal" user can switch between an ideal and a non-ideal LNA module. The IP3 point can be set with the parameter _ip3.



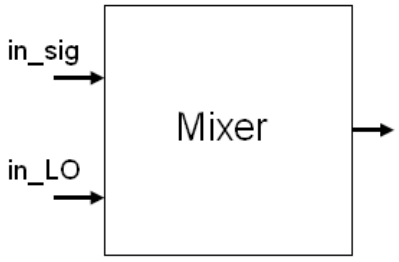
Class definition: lna(sc_core::sc_module_name n, double _gain, double _ip3, bool _ideal);

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
_gain	double	-	gain in dB
_ip3	double	-	IP3 in dBm
_ideal	bool	-	true for simulation of ideal LNA, otherwise false

2.10 Mixer

This class converts the input signal from low frequency to high frequency or vice versa. User can set the gain of mixer with the parameter _gain (default equals 1). This class also allows user to model the intercept modulation of mixer. With the parameter "_ideal" user can switch between an ideal and a non-ideal LNA module. The IP3 point can be set with the parameter _ip3.



Class definition: mixer(sc_core::sc_module_name n, double gain, double ip3, bool _ideal)

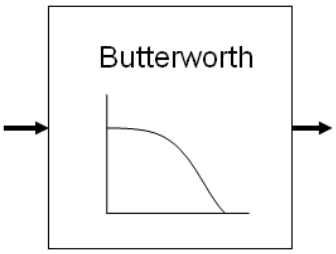
Interfaces: sca_tdf::sca_in<double> sig_in;
sca_tdf::sca_in<double> lo_in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
_gain	double	-	gain in dB
_ip3	double	-	IP3 in dBm
_ideal	bool	-	true for simulation of ideal mixer, otherwise false

2.11 Analog Filters

2.11.1 Butterwoth lowpass/highpass filter(frequency responce!!)

This class models the Butterworth lowpass and highpass filter. It is designed to have a frequency response which is as flat as mathematically possible in the passband.

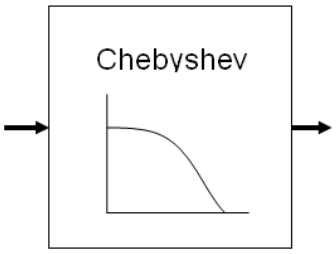


Class definition: btworth(sc_core::sc_module_name n,string _type, double _gp, double _gs, double _wp, double _ws);
Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_type	string	-	select between lowpass and highpass
_gp	double	-	passband gain in dB
_gs	double	-	stopband gain in dB
_wp	double	-	passband frequency in rad/s
_ws	double	-	stopband frequency in rad/s

2.11.2 Chebyshev lowpass/highpass filter

This class models the Chebyshev lowpass and highpass filter(Type 1) which has a steeper roll-off and more passband ripple than Butterworth filters.



Class definition:
chebyshev(sc_core::sc_module_name n, string _type="lowpass", double _ratio=2., double _gs=-20., double _wp=10, double _ws=20);

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_type	string	“lowpass”	select between “lowpass” and “highpass“
_ratio	double	2	**the ratio** of maximum gain to the minium gain in passpand **in dB**
_gs	double	20	stopband gain in dB
_wp	double	10	passband frequency in rad/s
_ws	double	20	stopband frequency in rad/s

2.12 45° Shifter

This class generates a phase shifting of +45° or -45° to input signals. With the help of parameter “delta_r” and “delta_c” it is possible to model the variation of resistor value and/or capacitor value which is usually caused by process variation.

Class definition:
shift_45(sc_core::sc_module_name n,string mode, double w_if, double delta_r=0, double delta_c=0);

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
mode	string	-	select “pos” for +45° shifting and “neg“ for -45° shifting
w_if	double	-	Frequency of input signal
delta_r	double	0.0	variance of resistor value in percentage respect to its ideal value
delta_c	double	0.0	variance of capacity value in

			percentage respect to its ideal value

2.13 Gaussian Wave-shaping filter

This class models gaussian pulse shaping which is used widely in digital communication systems to minimize the out of band spectral energy. The baseband rectangular pulse stream is passed through the model before frequency modulating the carrier.

Class definition:

gauss_shaping(sc_core::sc_module_name n, double _bts, double _ts);

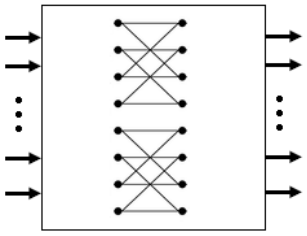
Interfaces:

sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out

Parameter	Type	Default value	Description
n	sc_module_name	-	
mode	string	-	select “pos” for +45° shifting and “neg“ for -45° shifting
_bts	double	-	product of bit period of the input stream and -3dB bandwidth of the Gaussian filter
_ts	double	-	bit period of the input stream

2.14 FFT/IFFT

This class models n-point forward and inverse fast Fourier transform. “n” can be set with template N, which is an positive integer number. The template defines at the same time the number of input and output ports. This class is built as a parallel-in and parallel-out module. The inputs of the class are separated into in_real and in_imag ports as well as out_real and out_imag on the output side. Please use in_real and out_real for I signls and in_imag and out_imag for Q signals respectively. In case there are only real signals (no Q signals) the in_imag ports should be connected with the ground.



Class definition:

template <int N>
fft_ifft(sc_core::sc_module_name n, string _mode)

Interfaces: sca_tdf::sca_in<double> in_real[N];
sca_tdf::sca_in<double> in_imag[N];

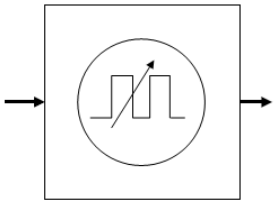
sca_tdf::sca_out<double> out_real[N];
sca_tdf::sca_out<double> out_imag[N];

Parameter	Type	Default value	Description
nm	sc_module_name	-	
_mode	string	-	select between “FFT” and “IFFT”

2.15 Phase-locked loop

2.15.1 Digital VCO

This class models a digital voltage controlled oscillator. It is used to generate a single-tone square wave with tunable frequency. Tuning is done using a contral voltage at the input ***in***. Without input voltage the oscillator runs at its free running frequency.



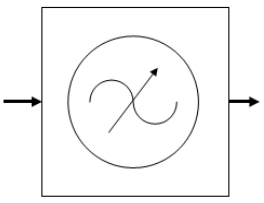
Class definition: d_vco(sc_core::sc_module_name n, double freq_data, int datarate, double _kvco, double _gain)

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
freq_data	double	-	central frequency
data_rate	integer	-	datarate of the output
_kvco	double	-	sensitivity
_gain	double	-	gain

2.15.2 Analog VCO

This class models a digital voltage controlled oscillator. It is used to generate a single-tone sine wave with tunable frequency. Tuning is done using a control voltage at the input **in**. Without input voltage the oscillator runs at its free running frequency.



Class definition: a_vco(sc_core::sc_module_name n, double freq_data, int datarate, double _kvco, double _gain)

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
freq_data	double	-	central frequency
data_rate	integer	-	datarate of the output
_kvco	double	-	sensitivity
_gain	double	-	amplitude of VCO output signal

2.15.3 Lowpass-filter

This class models a general first order lowpass filter. User can set the cutoff frequency of the filter with parameter freq_cut.

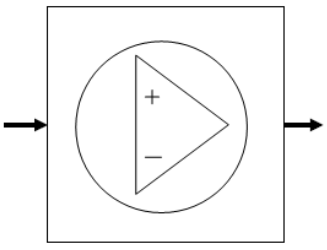
Class definition: lp(sc_core::sc_module_name n, double freq_cut)

Interfaces: sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
freq_cut	double	-	cutoff frequency of filter

2.15.4 Phase Detector (Comparator)

This class models a phase detector. It is a frequency mixer or analog multiplier that generates a voltage signal which represents the difference in phase between two signal inputs.



Class definition: phc(sc_core::sc_module_name n, int datarate, double _gain);

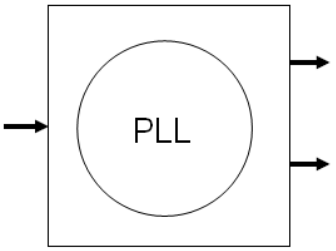
Interfaces: sca_tdf::sca_in<double> in_ref; // input port for reference signal
sca_tdf::sca_in<double> in_vco; // input port for VCO output signal
sca_tdf::sca_out<double> out; // output port

Parameter	Type	Default value	Description
n	sc_module_name	-	

data_rate	integer	-	datarate of the output
_gain	double	-	gain

2.15.5 PLL

This class generates a signal that has a fixed relation to the phase of a "reference" signal.



Class definition: pll(sc_core::sc_module_name n, double phc_gain, double lp_fc, double vco_freq, double kvco, double vco_gain)

Interfaces:
```
sca_tdf::sca_in<double>   ref;          //input port for reference signal
sca_tdf::sca_out<double>  vcoo; //output port of VCO
sca_tdf::sca_out<double>  lpo;          //output port of lowpass filter
```

Parameter	Type	Default value	Description
n	sc_module_name	-	
phc_gain	double	-	gain of phase detector
lp_fc	double	-	cut off frequency of LP filter
vco_freq	double	-	central frequency of VCO
kvco	double	-	sensitivity of vco
vco_gain	double	-	amplitude of VCO output signal

2.16 Channel

This class models the transmission environment. With parameter “atten” user can set the attenuation of the transmission channel. Parameter “n_art” defines the type of noise that will be modelled in the channel. At the moment user can switch between uniform distributed and additive Gaussian white noise. By using uniform distributed noise one can set the minimal and maximal value of the noise with parameter “a” and “b”. They define the variance and mean of the noise if additive Gaussian white noise is modelled.

Class definition:
channel (sc_core::sc_module_name n, double atten, string n_art,double a,double b, int d_rate);

Interfaces:
```
sca_tdf::sca_in<double> in;         // input double (wave)
sca_tdf::sca_out<double> out;
```

Parameter	Type	Default value	Description
nm	sc_module_name		
atten	double	-	attenuation of the channel
n_art	string	-	noise type, “uniform” and “gauss_white” are available at the moment
a	double	-	The parameter defines the minimal value of noise when “n_art” is set to “uniform” and defines the variance of noise if “n_art” is set to “gauss_white”
b	double	-	The parameter defines the maximal value of noise when “n_art” is set to “uniform” and defines the mean of noise if “n_art” is set to “gauss_white”
d_rate	int	-	data rate of input and output signal

2.17 Analyse tools

2.17.1 Eye Diagram

The eye diagram is a widely used plot format to evaluate the quality of digital modulated signals. The signal are plotted over a period of an integer multiple of the symbol duration.This functionality is realized in the waveform viewers of system level simulators as postprocessing.

Note: In the end of your program (in main.cpp) the method finish() must be called to generate the SVG-File and finish the file handling.

Class definition: eyediag(sc_core::sc_module_name n, double periodetime_, double sigamp_, int periodes_, int delay_, int in_rate)

Interfaces: sca_tdf::sca_in<double> in;

Parameter	Type	Default value	Description
n	sc_module_name	-	
periodetime_	double	-	symbol duration of original sigal
sigamp_	double	-	maximal possible amplitude of signal to be analysed
periodes_	integer	-	number of periods to be plotted
delay_	integer	-	number of periods to be ignored before plot
in_rate	integer	10	data rate of input port

2.17.2 Scatter Plot

This class plots the scatter diagram of input signal.

Class definition: scatter(sc_core::sc_module_name n, double sigamp_, int in_rate)

Interfaces: sca_tdf::sca_in<double> in;

Parameter	Type	Default value	Description
n	sc_module_name	-	
sigamp_	double	-	maximal possible amplitude of signal to be analysed
in_rate	integer	10	data rate of input port

2.17.3 Network analyzer

This class models a network analyzer which is used to analyze the frequency response of electrical networks.

Class definition: nwa(sc_core::sc_module_name n, double _amplitude, double _start_f, double _stop_f, double _step_f, double _min_dB, double _max_dB, double _period_number)

Interfaces:

sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;

Parameter	Type	Default value	Description
n	sc_module_name	-	
_amplitude	double	-	amplitude of mess signal
_start_f	double	-	start frequency of mess signal in Hz
_stop_f	double	-	stop frequency of mess signal in Hz
_step_f	double	-	frequency step
_min_dB	double	-	illustratable minimal value of mess result
_max_dB	double	--	illustratable maximal value of mess result
_period_number	double		number of periods of mess signal with start frequency

3 Example: OFDM Transmitter system

The purpose of this chapter is to show the user how to use the modules from the TU Vienna SystemC AMS communications library to model communication systems. We assume that the user who uses this library already has some basic knowledge of SystemC AMS.

Based on the previously described modules an OFDM transmitter system is built up (see Figure 1) in this Section.

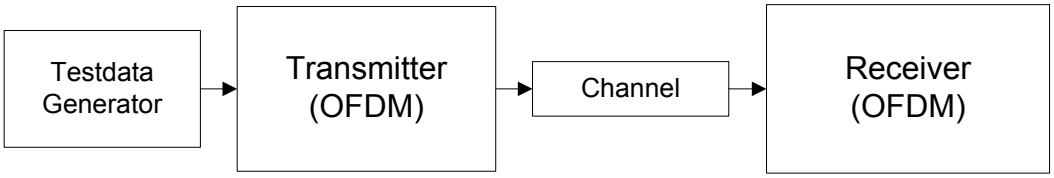


Figure 1 Top level block diagram of the example application

Figure 1 shows the structure of the transmitter system in our example. First, test data is generated by a test generator and passed to the transmitter. As the test generator the **rand_bool** module described in Section 2.1.3 is used and a serial of universal distributed random bits is generated. The transmitter takes these binary signals and modulates them to a high frequency OFDM signal (using Orthogonal Frequency Division Multiplex (OFDM) modulation). The **ofdm_se** module described in Section 2.2.20 is used to model the transmitter. The generated signal is then passed via a channel (modelled using the **air** module described in section 2.16) which attenuates its input signal and adds noise. After that the signal is taken by a receiver and translated back into a stream of binary bits.

In order to model communication systems using the modules in the library the user has to include the respective header data and set the right namespace first:

```
#include "tuv_ams_library/tuv_ams_library.h"
using namespace TUV_ams_lib::bb;
```

Then the expected modules from the library have to be instantiated in the following way:

```
rand_bool i_stimuli("stimuli",16);
```

where “rand_bool” is the module name, “i_stimuli” is the instance name of the module, “stimuli” and “16” are values of corresponding parameters of the module.

Finally, SystemC AMS signals have to be declared to connect different modules:

```
sca_tdf::sca_signal<bool> sig_stimuli;
i_stimuli.out(sig_stimuli);
```

Here a signal called “sig_stimuli” is declared and connected to the “out” port of the module “i_stimuli”.

Apart from the above mentioned issues the user has to set the time resolution of the simulation using the predefined method “sc_set_time_resolution()”. It is also required to set the sampling rate on at least one port of the modelled system with the method “.set_timestep()”.

For the system described above we need the following section of codes:

```
int sc_main(int argc, char* argv[])
{
  sc_set_time_resolution(1, SC_PS);    // set the time resolution

  sca_tdf::sca_signal<bool> sig_stimuli;    // declare SystemC AMS signals
  sca_tdf::sca_signal<double> sig_out;
  sca_tdf::sca_signal<double> noise_out;
  ...

  rand_bool i_stimuli("stimuli",16);   // instantiate test generator
  ofdm_se<8> i_tran("transmitter",freq_carrier,constl_dim,freq_bit,data_rate,
            ampl_se);                  // instantiate transmitter
  air i_air("air",attent,"gauss_white",n_va,n_mean,data_rate);
                                       // instantiate channel
  ofdm_re<8> i_receiver("receiver",freq_carrier,constl_dim,freq_bit,data_rate,
            ampl_re);                  // instantiate receiver
  drain drn("drn");                    // instantiate drain. This is only a
                                       // module used to consume tokens as we
                                       // can not let a SystemC AMS scheduling
                                       // loop open.

  i_stimuli.out.set_timestep(1/freq_bit,SC_SEC); //Time step has to be set to
  one of the ports in the system

  i_stimuli.out(sig_stimuli);          // connect different modules
  i_tran.in(sig_stimuli);
  i_tran.out(sig_out);
  ...

  sca_util::sca_trace_file wave = sca_util::sca_create_vcd_trace_file ("wave");
// VCD trace file:
  sca_util::sca_trace (wave, sig_stimuli ,"stimuli");
  ...

  sc_start(0.05,SC_MS);                // Simulation time: 0.05ms
  return 0;
}
```

(This is not the complete source code)

The transmitter and receiver modules consist again of sub modules, which are contained in the library. Figure 2 and Figure 3 present the internal structure of the transmitter module and receiver module, respectively.

The transmitter takes a serial stream of binary digits. By inverse multiplexing, these are first demultiplexed into N parallel streams, and each one mapped to a (possibly complex) symbol stream using QAM modulation.

An inverse FFT is computed on each set of symbols, giving a set of complex time-domain samples. These samples are then quadrature-mixed to passband in the standard way.

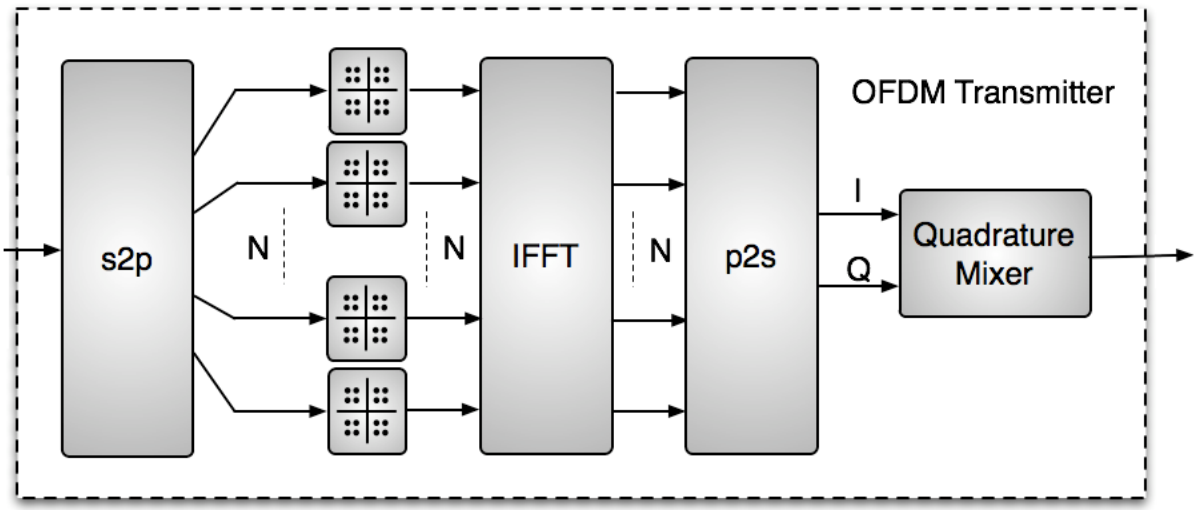


Figure 2 Block diagram of OFDM transmitter

The receiver picks up the signal from antenna, which is then quadrature-mixed down to baseband using cosine and sine waves at the carrier frequency. This also creates signals centered on $2*f_{carrier}$, so low-pass filters are used to reject these. The baseband signals are then sampled and a forward FFT is used to convert back to the frequency domain. This returns N parallel streams, each of which is converted to a binary stream using an appropriate symbol detector. These streams are then re-combined into a serial stream, which is an estimate of the original binary stream at the transmitter.

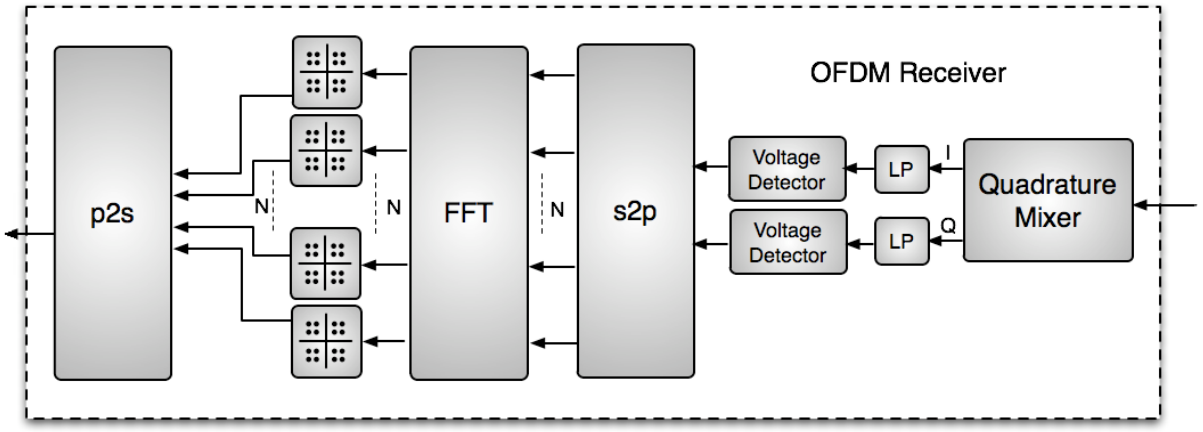


Figure 3 Block diagram of OFDM Receiver

The next code Section shows how to use building block modules to build the OFDM transmitter module:

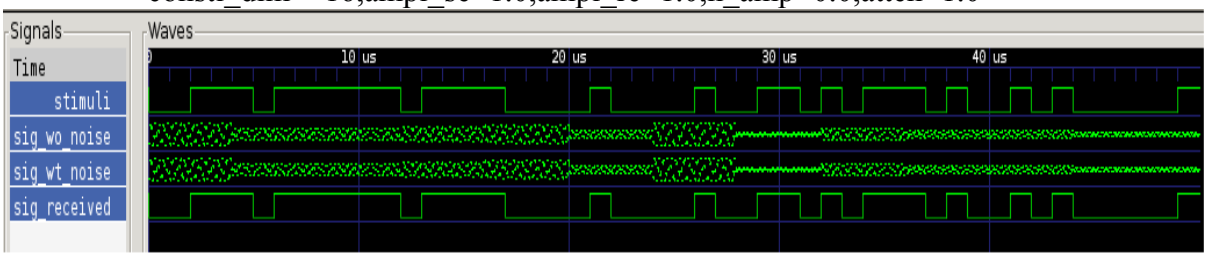
```
template <int N>
SC_MODULE(ofdm_se){
  public:
   //Ports:
   sca_tdf::sca_in<bool> in;            // declare port
   sca_tdf::sca_out<double> out;
   //signals
   sca_tdf::sca_signal<bool> sig_pa[N];      // declare signals to
   sca_tdf::sca_signal<double> sig_real[N];  // connect sub modules
   ...
  private:
  //module instantiation
  s2p<bool,N>* s2p_sub;
  qam_map* qam_mapper_sub[N];
  fft_ifft<N>* ifft_sub;
  p2s<double,N>* p2s_r_sub;
  p2s<double,N>* p2s_i_sub;
  q_mixer_tr* q_mixer_tr_sub;
 public:
  //Constructor
  ofdm_se(sc_module_name n, double mixer_fc,int qam_p_num, double bit_f,int
         dout_rate,double _amp=1)
  {
     int mixer_rate=(int)floor(dout_rate*log2(qam_p_num)*mixer_fc/bit_f);
  // instantiating modules and connecting them using signals
     s2p_sub = new s2p<bool,N>("s2p_sub",1);  // serial to parallel module
     s2p_sub->in(in);
     for(int i=0;i<N;i++)
       s2p_sub->out[i](sig_pa[i]);

      ...

      ifft_sub = new fft_ifft<N>("ifft_sub","IFFT"); //IFFT module
      for(int i=0;i<N;i++)
        {
         ifft_sub->in_real[i](sig_real[i]);
         ifft_sub->in_imag[i](sig_imag[i]);
         ifft_sub->out_real[i](sig_out_real[i]);
         ifft_sub->out_imag[i](sig_out_imag[i]);
        }

      ...

      p2s_r_sub = new p2s<double,N>("p2s_r_sub",1); //quadrature mixer module
      q_mixer_tr_sub = new
      q_mixer_tr("q_mixer_tr_sub",mixer_fc,_amp,0.,0.,mixer_rate,false);
      q_mixer_tr_sub->i_in(sig_out_i);
      q_mixer_tr_sub->q_in(sig_out_q);
      q_mixer_tr_sub->out(out); /***/

    }
  };
```

As we can see from the source code the modelling of an OFDM transmitter is quite easy when using the existing modules in the library.

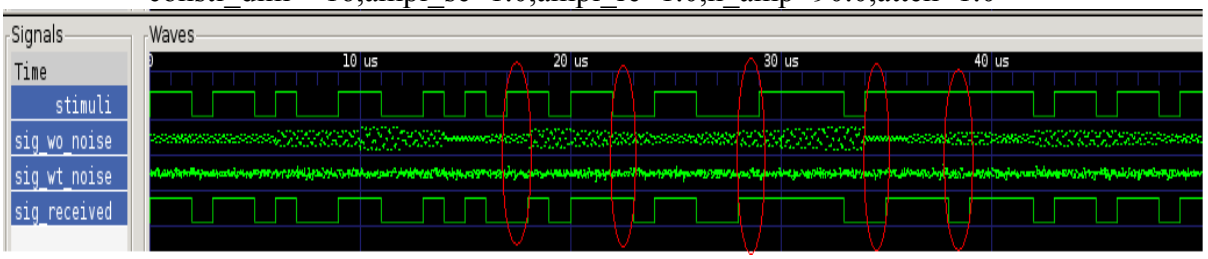
Simulation example:
In this Section several simulation results are presented with different settings of the following parameters:

- constl_dim, defines the number of points in the QAM constellation
- ampl_se , defines the amplitude of the carrier for the transmitter
- ampl_re, defines the amplitude of the carrier for the receiver
- n_amp, maximal value of noise in channel
- atten, attenuation of channel

1. without noise and attenuation in channel:
constl_dim = 16,ampl_se=1.0,ampl_re=1.0,n_amp=0.0,atten=1.0

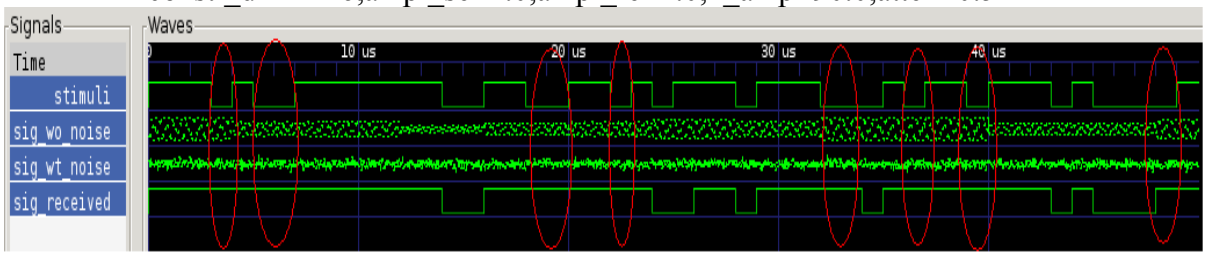


2. with gaussian noise ,without attenuation in channel:
constl_dim = 16,ampl_se=1.0,ampl_re=1.0,n_amp=90.0,atten=1.0



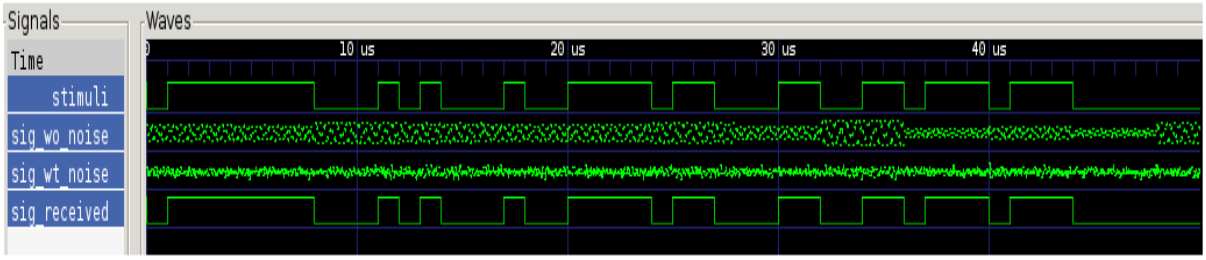
As we can see that because of the noise in the channel we got 5 error bits.

3. with gaussian noise and 50% attenuation in channel:
constl_dim = 16,ampl_se=1.0,ampl_re=1.0,n_amp=90.0,atten=0.5



Having both noise and attenuation in the channel the number of error bits became larger.

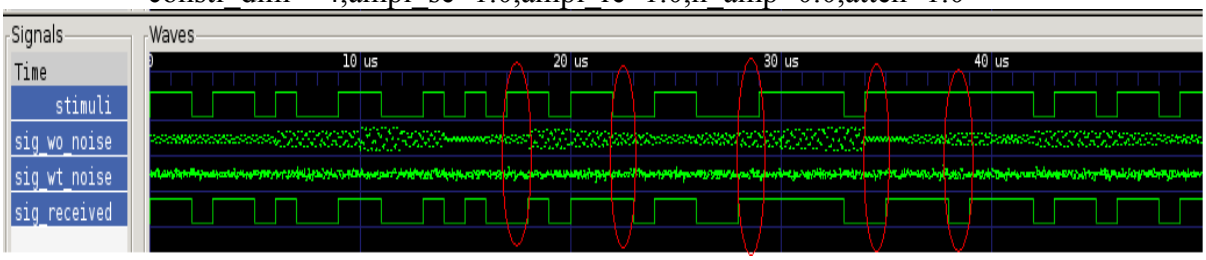
4. with gaussian noise and 50% attenuation, but with higher transmission power
constl_dim = 16,ampl_se=20,ampl_re=0.1,n_amp=90.0,atten=1.0



As presented in this figure we can reproduce the correct signals with higher transmission power. This is due to the parameter “ampl_se”, which is set to 20 and means the signal from the transmitter is 20 times as strong as by previous simulations.

5. with gaussian noise and 50% attenuation, but slowing down transmission speed

constl_dim = 4,ampl_se=1.0,ampl_re=1.0,n_amp=0.0,atten=1.0



Or it is also possible to slow down the transmission speed to improve the BER. This is due to the parameter “constl_dim” set to 4, which means now only 2 bits (which is 4 bits when setting “constl_dim” to 16) were encoded at one time.