Example: QPSK Transmitter System

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

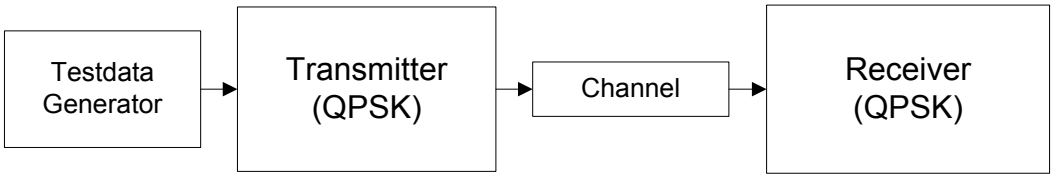


Figure 1 Top level block diagram of the QPSK Transmitter and Receiver

Figure 1 shows the Functional block diagram of QPSK Transmitter and Receiver system. First, test data is generated by a test generator and passed to the transmitter. The **rand_bool** module is used test data generator and a serial of universal distributed random bits are is generated. The transmitter takes these binary signals and modulates those bits to a high frequency QPSK signal (using Quadrature Phase Shift Keying modulation). The **qpsk (QPSK Modulator)** module as described in previous Section is used to model the transmitter. The generated signal is then passed via a channel (modelled using the **air** module) 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 "directory of the library/lib_v_01_11/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 binary_source ("stimuli", rate);
```

Where “rand_bool” is the module name, “binary_source” is the instance name of the module, “stimuli” and “rate” is the output data rate of the module.

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

```
sca_tdf::sca_signal<bool> binary_data_tx;
 binary_source.out (binary_data_tx);
```

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

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_T()”.

Following Section of code describe the System, mentioned above.

```
int sc_main(int argc, char* argv[])
{
  sc_set_time_resolution(1, SC_PS);

 /********** defining signals and parameters ********** */

  sca_tdf::sca_signal<bool> binary_data_tx;
  sca_tdf::sca_signal<double> modulated_data_tx;
  sca_tdf::sca_signal<double> modulated_data_noisy_tx;
  sca_tdf::sca_signal<bool> binary_data_rx;


 double freq;
  int rate;


  /********** setting parameters for simulation ********** */

  cout <<"\n" << "frequency= ";  cin >> freq;
  cout <<"\n";

  cout << "sample rate ="; cin >> rate;
    cout <<"\n";

  /********** instantiating SDF-modules**********/

  rand_bool binary_source("stimuli",rate);
  binary_source.out(binary_data_tx);
  binary_source.out.set_timestep(0.1,SC_MS);

  qpsk qpsk_tr("qpsk_tr",freq, rate);
  qpsk_tr.in(binary_data_tx);
  qpsk_tr.out(modulated_data_tx);

  air channel("air",0.4,"gauss_white",1,0,rate);
  channel.in(modulated_data_tx);
  channel.out(modulated_data_noisy_tx);
```

```
    qpsk_de qpsk_rx("qpsk_rx",freq,rate);
    qpsk_rx.in(modulated_data_noisy_tx);
    qpsk_rx.out(binary_data_rx);

    drain drn("drn");
    drn.in(binary_data_rx);


    /********* tracing of signals ***************************/
    sca_util::sca_trace_file* atf = sca_util::sca_create_vcd_trace_file( "tr" );


    sca_util::sca_trace( atf, binary_data_tx ,"binary_data_tx" );
    sca_util::sca_trace( atf, modulated_data_tx ,"modulated_data_tx" );
    sca_util::sca_trace( atf, modulated_data_noisy_tx ,"modulated_data_noisy_tx" );
    sca_util::sca_trace( atf, binary_data_rx,"binary_data_rx" );

    sc_start(20, SC_MS);

    sca_util::sca_close_vcd_trace_file( atf );

    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 de-multiplexed into N parallel streams, and each one mapped to a symbol stream using QPSK modulation.

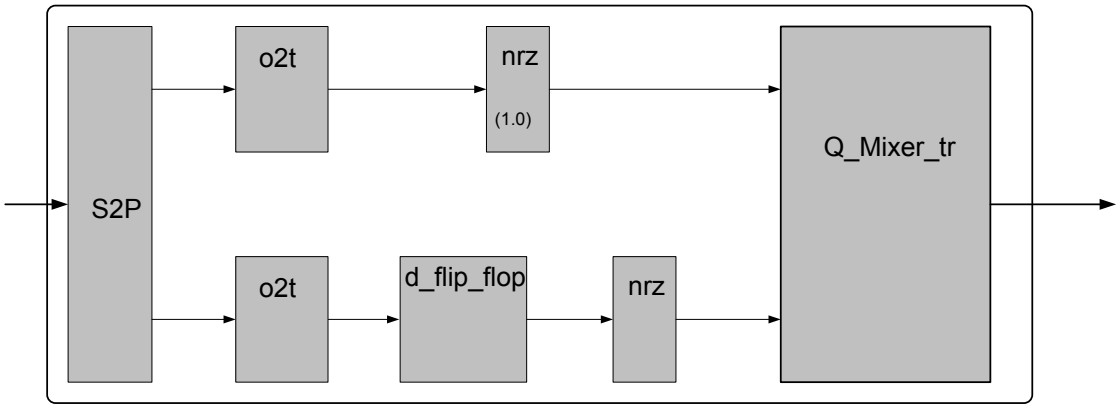


Figure 2 Block diagram of QPSK transmitter

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 2fc, so low-pass filters are used to reject these. The baseband signals are then sampled. 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.

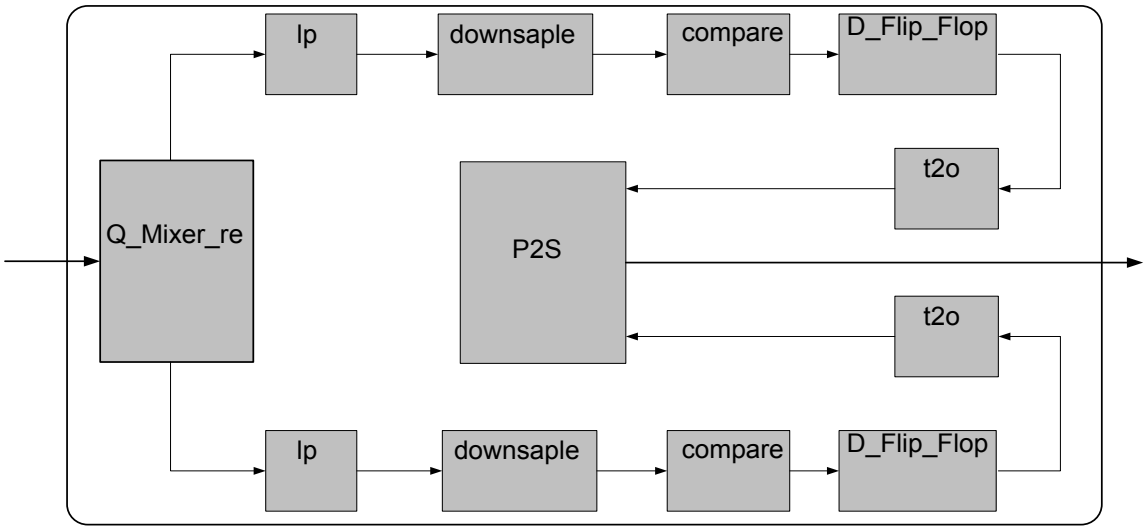


Figure 3 Block diagram of OFDM Receiver

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

```
  /************************* QPSK modulator*******************************/

SC_MODULE(qpsk) {

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

/***********signal for connecting sub module*******************/

 private:

sca_tdf::sca_signal<bool> sig_dcode;
sca_tdf::sca_signal<bool> sig_i;
sca_tdf::sca_signal<bool> sig_q;
sca_tdf::sca_signal<double> sig_n_i;
sca_tdf::sca_signal<double> sig_n_q;

/***************declare sub module*****************/
  s2p<bool,2>* s2p_sub;
  nrz* nrz_i_sub;
  nrz* nrz_q_sub;
  q_mixer_tr* mixer_sub;

 public:
    qpsk(sc_core::sc_module_name n, double _freq, int rate);
};

/***************************** r*******************************/
qpsk::qpsk(sc_core::sc_module_name n, double _freq, int rate)
    {
      s2p_sub = new s2p<bool,2>("i_s2p",1);
      s2p_sub->in(in);
      s2p_sub->out[0](sig_i);
      s2p_sub->out[1](sig_q);

      nrz_i_sub = new nrz("i_sub",1.0);
      nrz_i_sub->in(sig_i);
      nrz_i_sub->out(sig_n_i);

      nrz_q_sub = new nrz("q_sub",1.0);
      nrz_q_sub->in(sig_q);
      nrz_q_sub->out(sig_n_q);

      mixer_sub = new q_mixer_tr("i_mix",_freq,1.0,rate,false);
      mixer_sub -> i_in(sig_n_i);
      mixer_sub -> q_in(sig_n_q);
      mixer_sub -> out(out);
    }
```

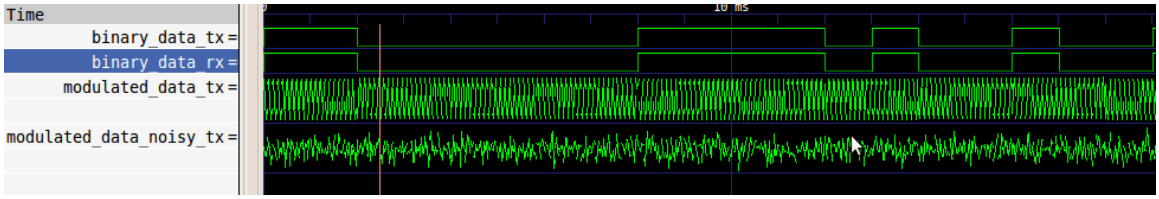
As we can see from the source code the modelling of an QPSK transmitter is quite easy when using the existing modules in the library.

Simulation example:

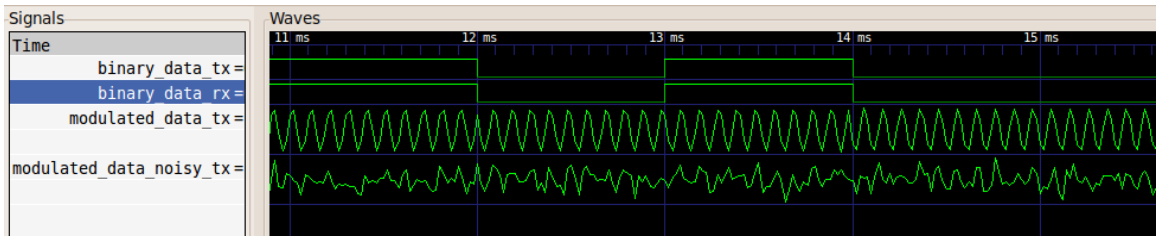
In this Section several simulation results are presented with different settings of the following parameters:

- freq, defines carrier frequency
- timestep (Baud Rate) , defines the Baud rate of Binary source
- atten, attenuation of channel

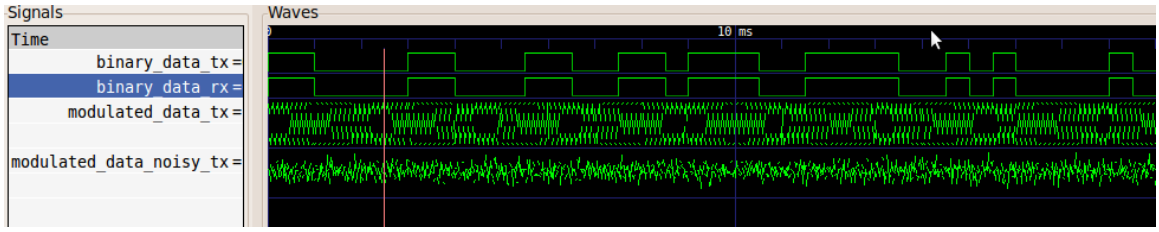
1. with Gaussian noise ,with attenuation in channel:
   freq = 10 K Hz, Baud Rate= I K, atten= 0.7



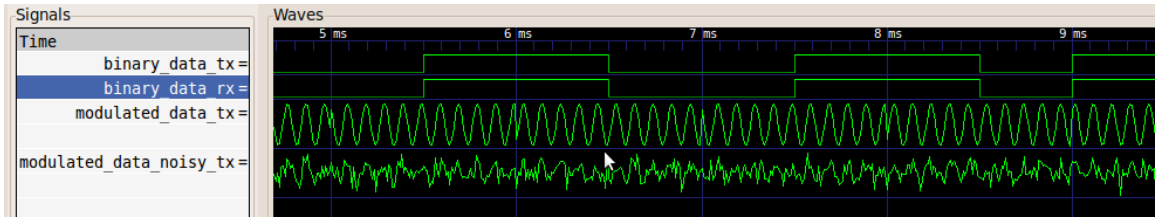
2. Above simulation with zoom in



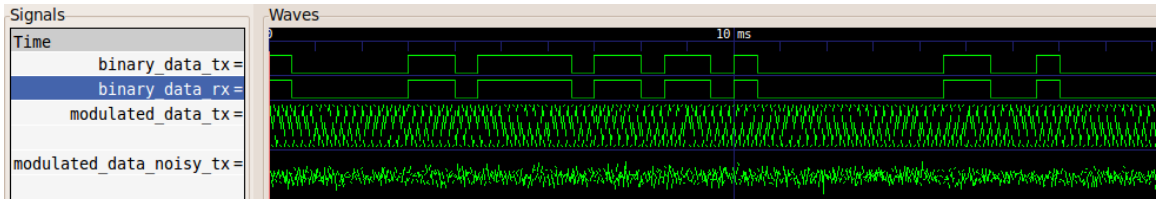
3. with Gaussian noise ,with attenuation in channel:
   freq = 10 K Hz, Baud Rate= 2 K, atten= 0.7



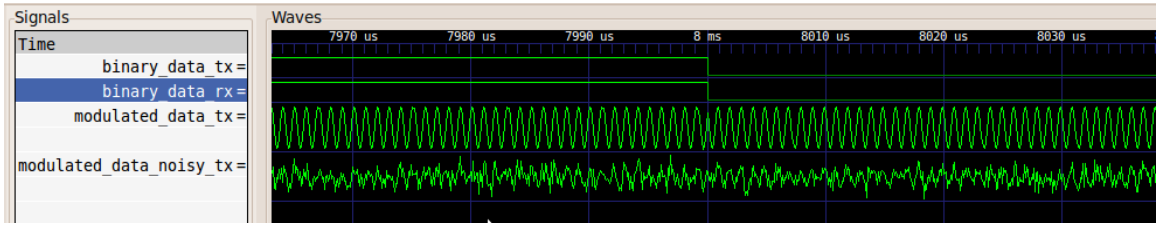
4. Above simulation with zoom in



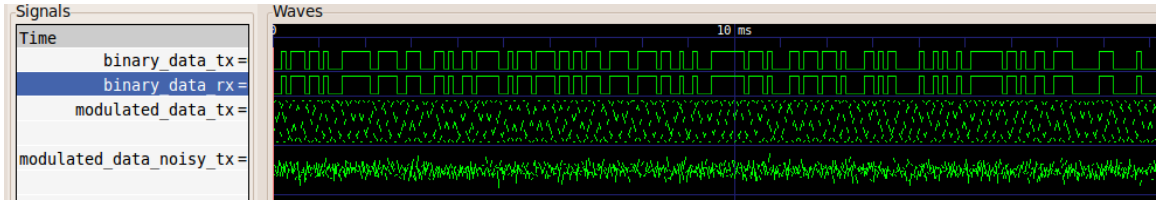
5. with Gaussian noise ,with attenuation in channel:
   freq = 1 M Hz, Baud Rate= 2 K, atten= 0.7



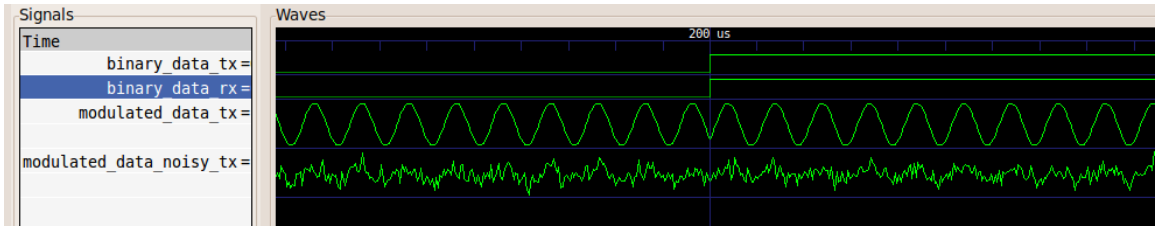
6. Above simulation with zoom in



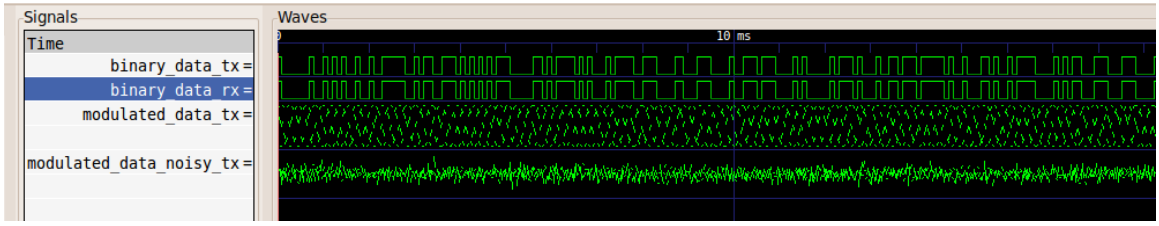
7. with Gaussian noise ,with attenuation in channel:
   freq = 1 M Hz, Baud Rate= 10 K, atten= 0.7



8. Above simulation with zoom in



9. with Gaussian noise ,with attenuation in channel:
   freq = 1 M Hz, Baud Rate= 10 K, atten= 0.5



10. with Gaussian noise ,with attenuation in channel:
11. freq = 1 M Hz, Baud Rate= 10 K, atten= 0.47