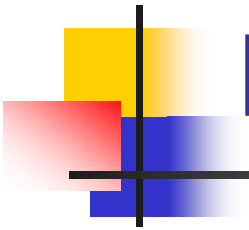
Modeling Heterogeneous Systems Using SystemC-AMS, A Case Study: A Wireless Sensor Network Node

{Michel.Vasilevski,François.Pecheux,
Hassan.Aboushady, Laurent.Delamarre}@lip6.fr

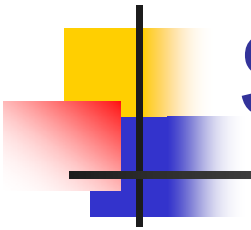
Laboratoire d'Informatique de Paris 6,
4, place Jussieu
75252 PARIS Cedex 05



Presentation overview

- SystemC-AMS, Models of Computation
- SystemC-AMS description of the WSN
- Results
- Ongoing research

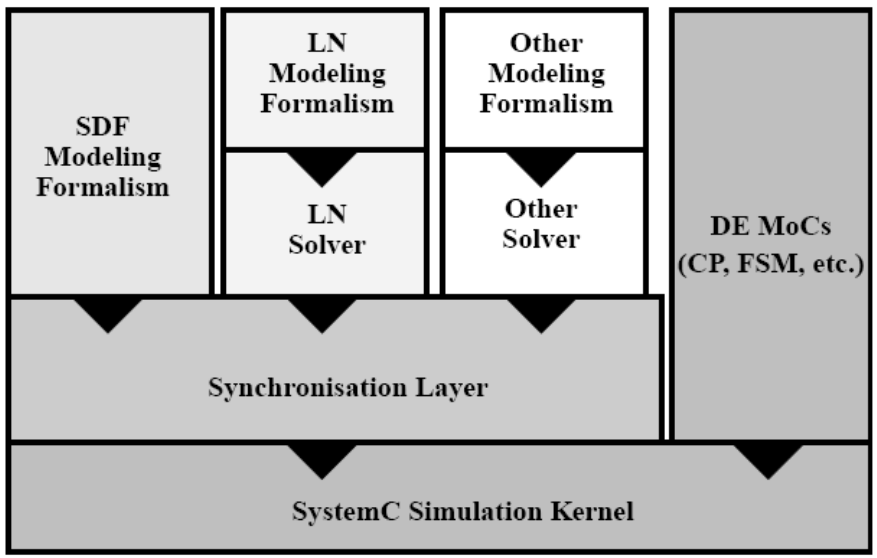
SystemC-AMS



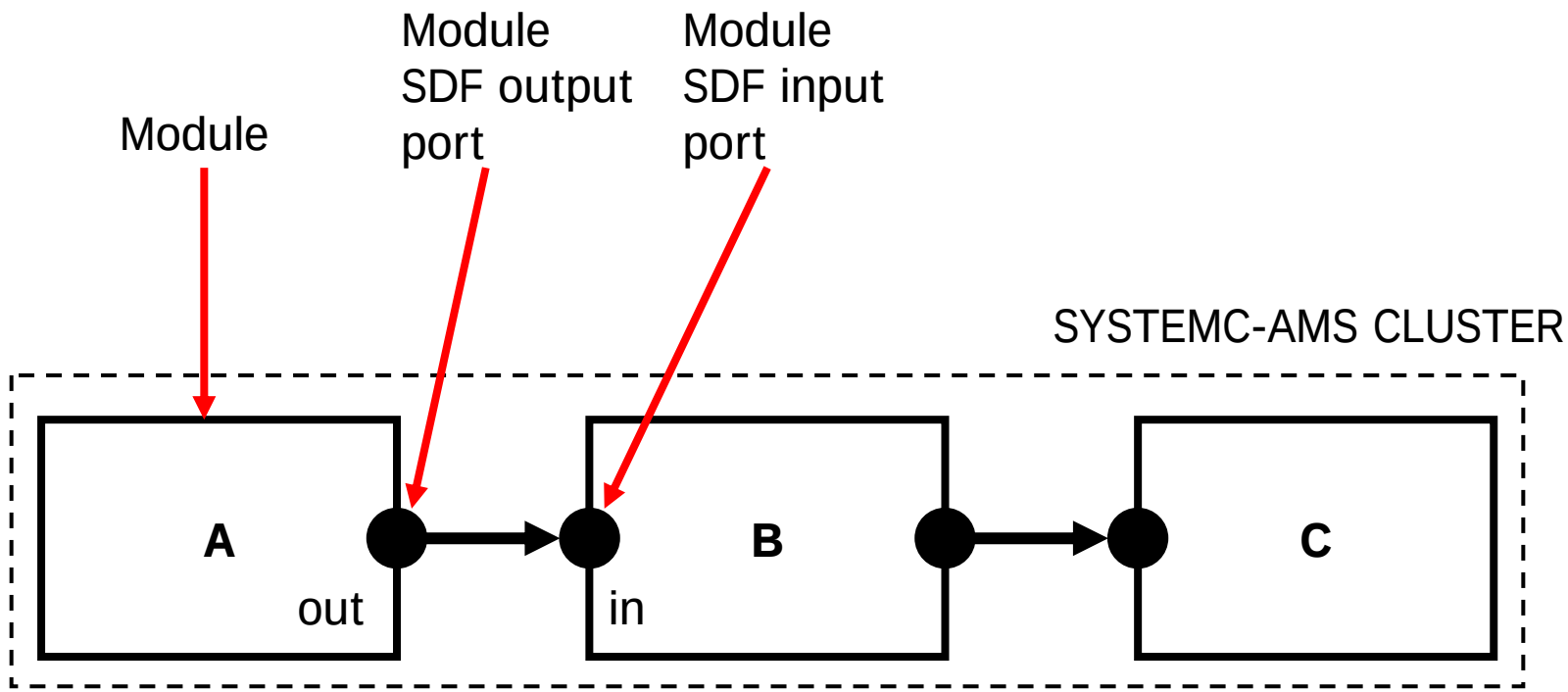
- Introduced by **Christoph Grimm** (TU Vienna), **Alain Vachoux** (EPFL), **Karsten Einwich** (Fraunhofer Institute Gesellschaft, Dresden)
- First prototype released by **Karsten Einwich**, available at www.systemc-ams.org, release candidate RC2
- **OSCI AMS** Working Group, managed by **Martin Barnasconi** (NXP). Lots of companies and academic partners
- Currently writing the LRM

2 Models of Computation implemented

- Conservative MoC
  - Dedicated to linear electrical networks, Kirchoff current law
  - If we limit application field to linear dynamic and non linear static, fast equation solver can be used
- Non-conservative MoC, Multi-rate Synchronous Dataflow



SDF Cluster

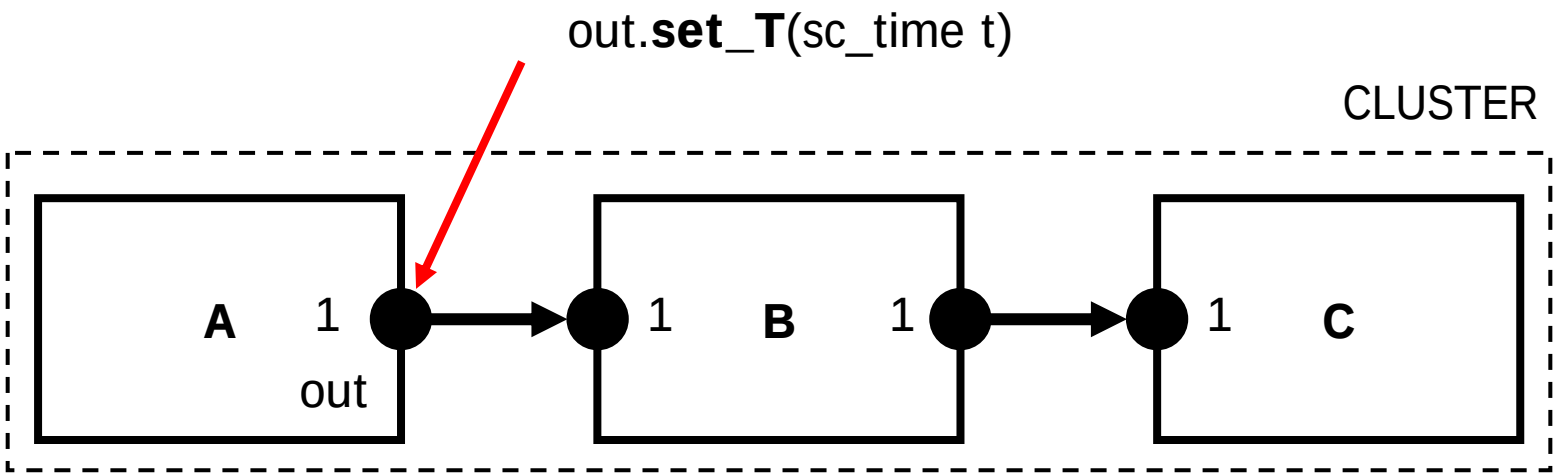

Module
Module
SDF output
port
Module
SDF input
port
SYSTEMC-AMS CLUSTER
A
out
B
in
C

Module behavior: **`sig_proc()`** function

```
sca_sdf_in<double> input;
sca_sdf_out<double> output;

void sig_proc()
{
...
output.write( BehaviorFunction(input.read(),...))
...
}
```

Single rate dataflow

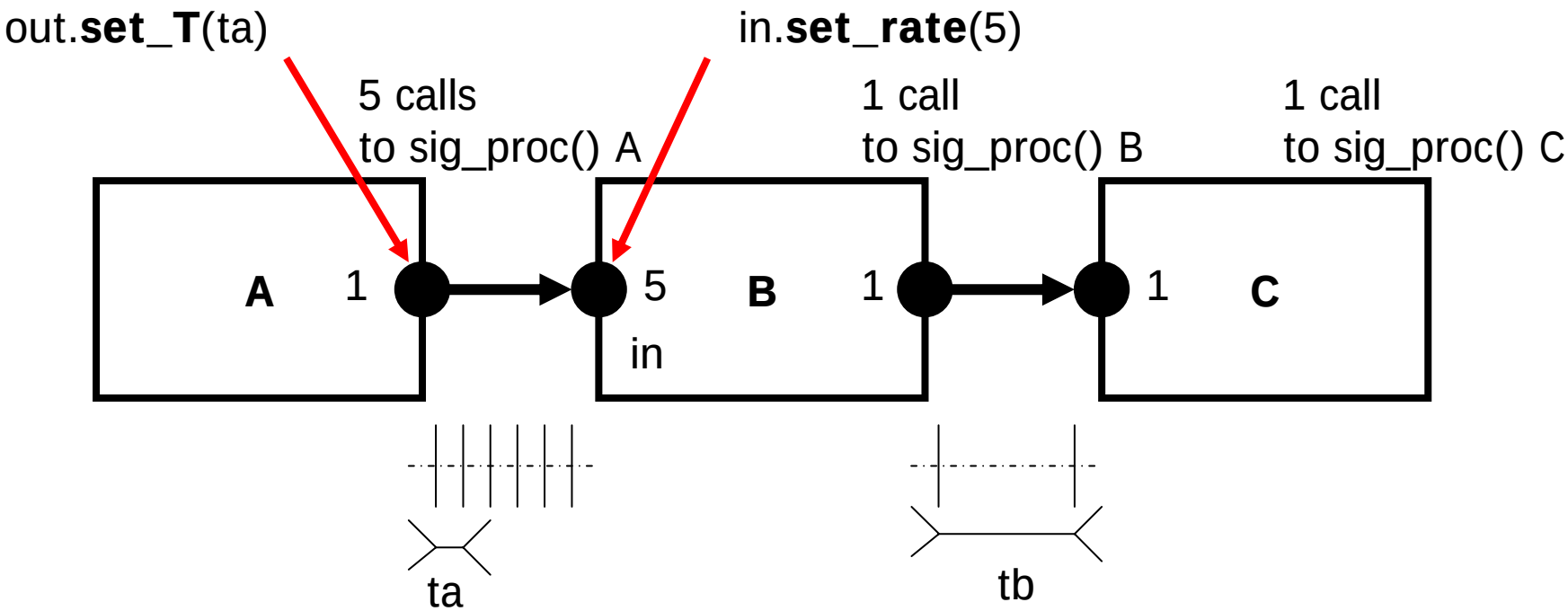
- Modules static scheduling computed during system elaboration
- A, B, and C are scheduled in that order when cluster is triggered
- Cluster sample period set with **set_T()**, propagated to all modules



Handling multirate dataflow

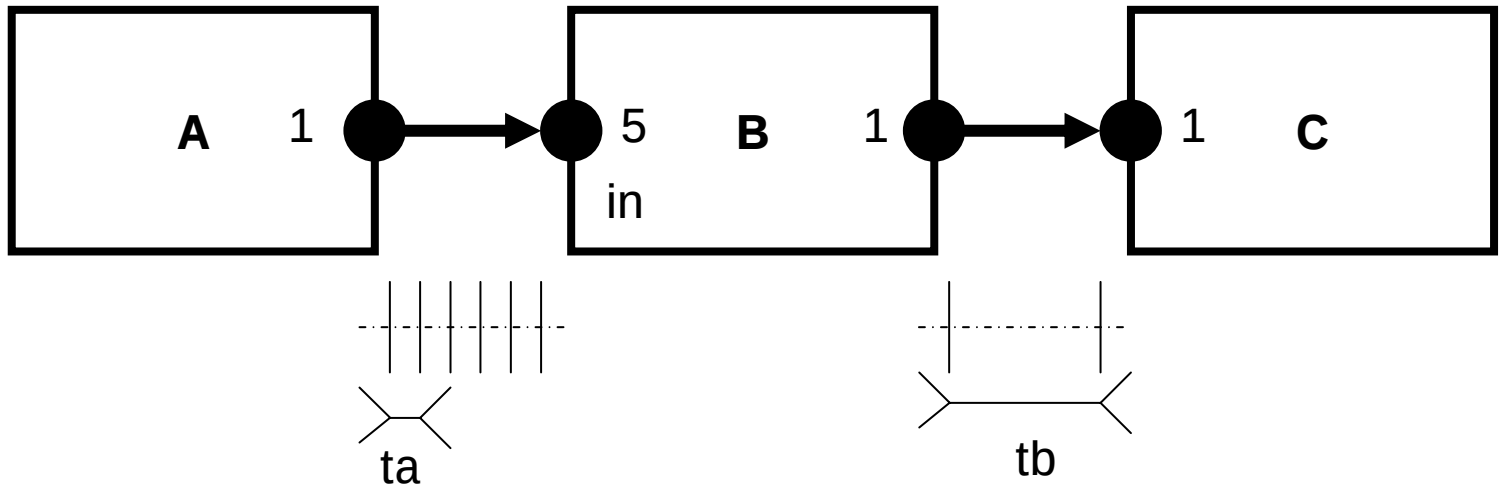
- **set_rate**() can be used on module ports to modify input (consuming) and output (producing) rates according to equation:

$$T_{in} * Rate_{in} = T_{out} * Rate_{out}$$

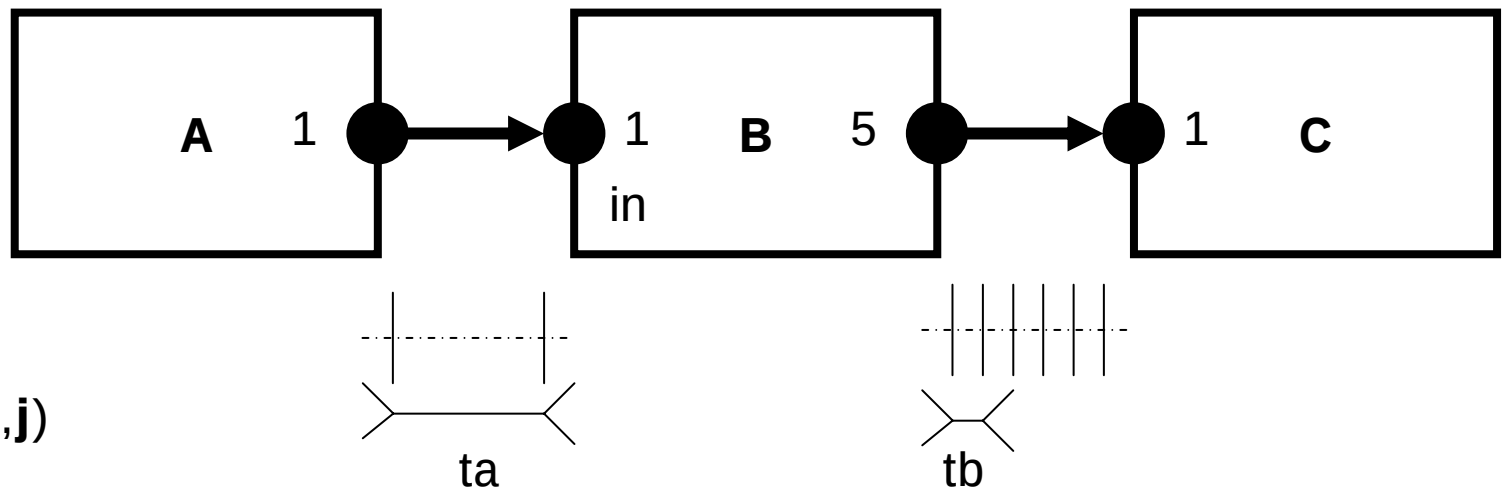


Decimator, interpolator

Decimator



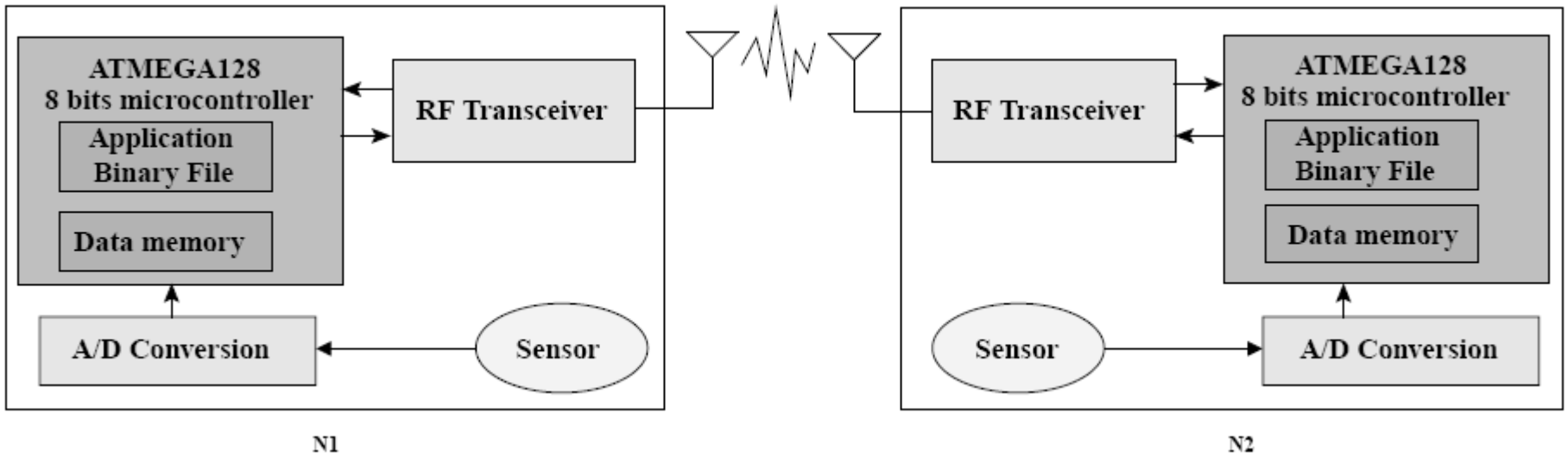
Interpolator



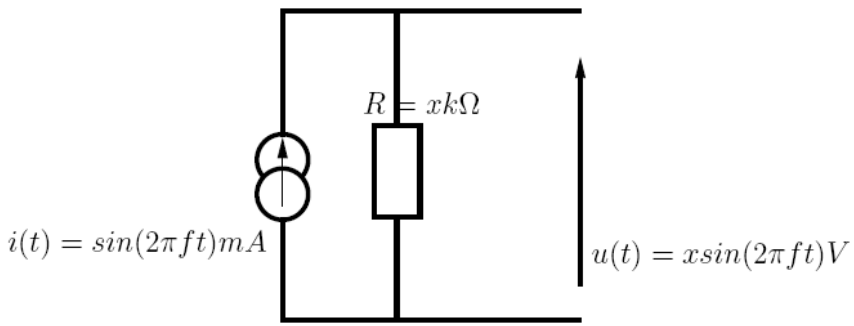
in.read(**i**)

out.write(val,**j**)

The modeled WSN



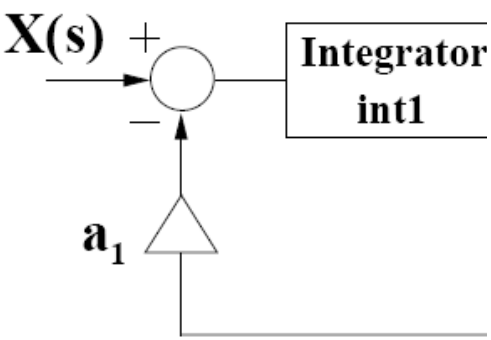
The sensor

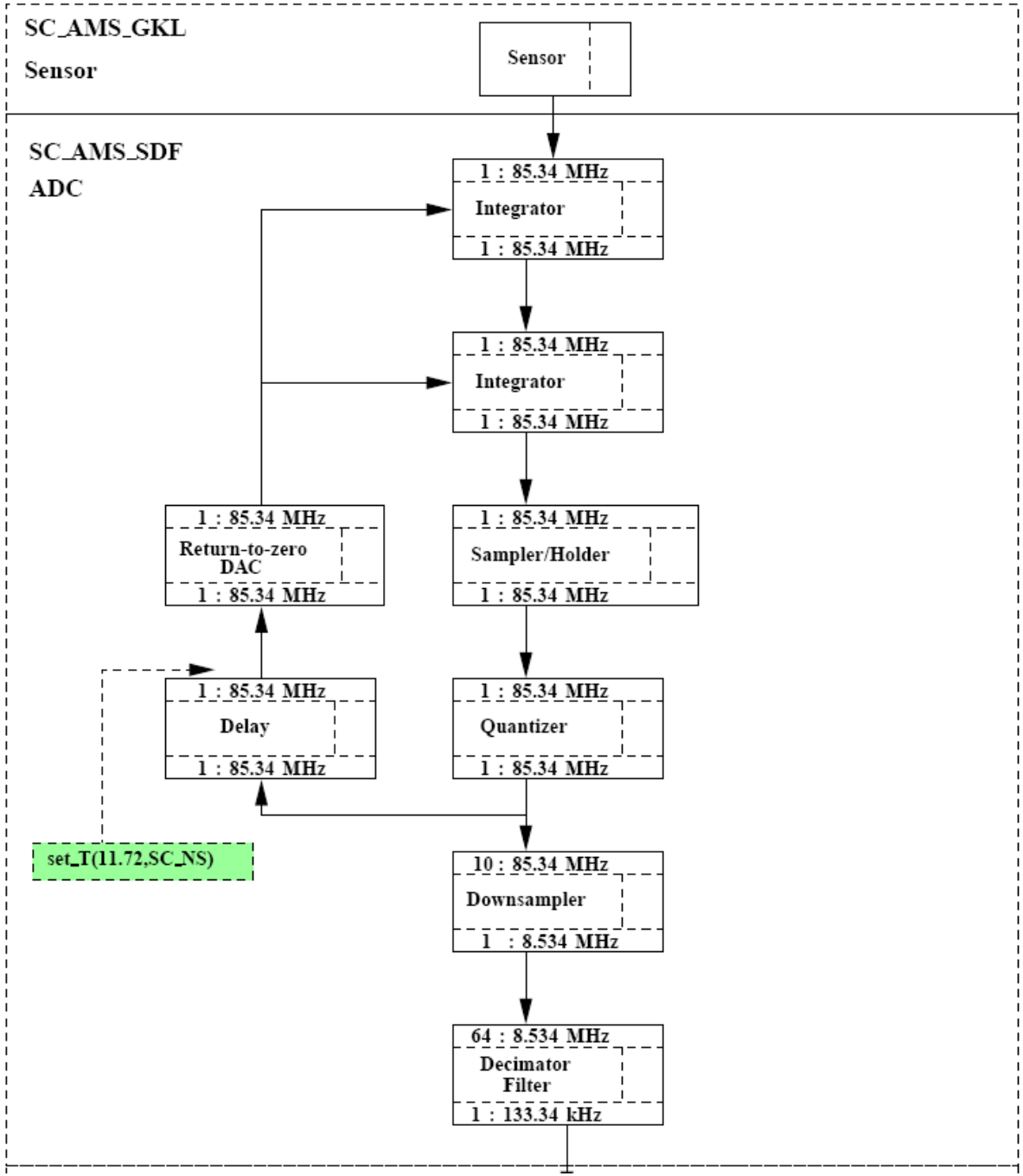



```
SC_MODULE (interface_v2sdf)
{
  sca_elec_port in;
  sca_sdf_out<double> out;

  sca_v2sdf *i_v2sdf_1;

  SC_CTOR (interface_v2sdf) {
    i_v2sdf_1 = new sca_v2sdf("i_v2sdf_1");
    i_v2sdf_1->p(in); // pos
    i_v2sdf_1->sdf_voltage(out); //
    i_v2sdf_1->scale=1.0; // scaling factor
  }
};
```

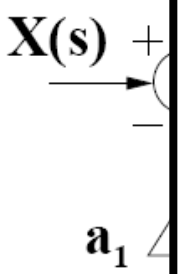
```
#ifndef WAVE_H
#define WAVE_H

SC_MODULE (wave)
{
  sca_elec_port w1;
  sca_elec_ref gnd;

  sca_isin  *i_sin_1;
  sca_r     *i_r_1;

  void init(double a, double f){
    i_r_1->value = a*1000;
    i_sin_1->freq = f;
  }

  SC_CTOR (wave) {
    i_sin_1=new sca_isin("i_sin_1");
    i_sin_1->p(w1); // pos
    i_sin_1->n(gnd); // neg
    i_sin_1->ampl=0.001; // magnitude in A

    i_r_1=new sca_r("r1");
    i_r_1->p(w1);
    i_r_1->n(gnd);
  }
};
#endif
```

2nd orde convert





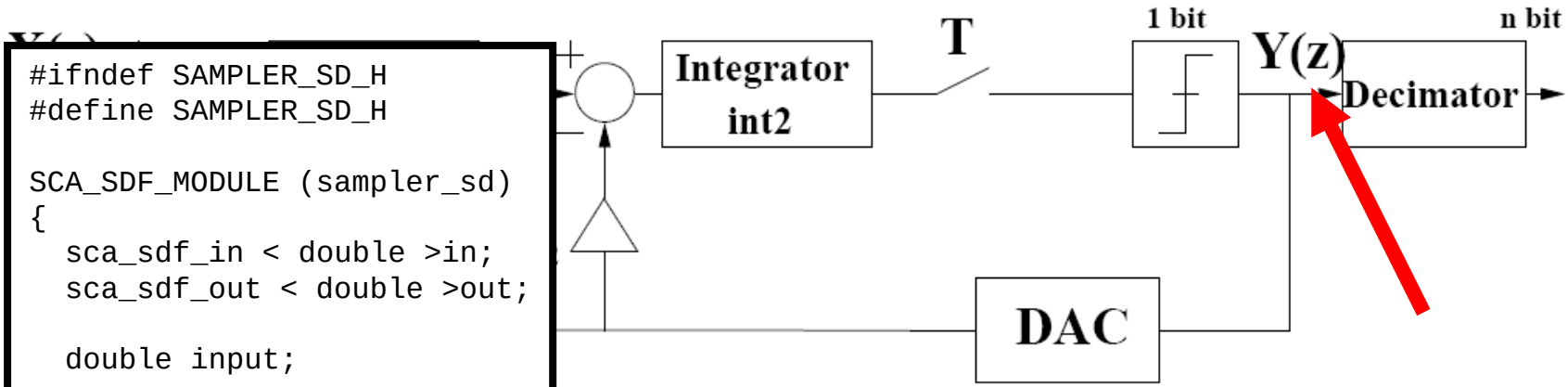
2nd order Sigma-Delta Integrator




```
#ifndef INTEGRATOR_SD_H
#define INTEGRATOR_SD_H

SCA_SDF_MODULE (integrator_sd)
{
  sca_sdf_in < double >in1;
  sca_sdf_in < double >in2;
  sca_sdf_out < double >out;

  double fs, ai, ki;
  sca_vector < double >NUM, DEN, S;
  sca_ltf_nd ltf1;

  void init (double a, double f, double k) {
    DEN (0) = 0.0;
    DEN (1) = 1.0;
    NUM (0) = 1.0/3.0;
    ai=a;
    fs=f;
    ki=k;
  }

  void sig_proc () {
    out.write (ltf1(NUM, DEN, S, fs * (ai * in1.read () - ki * in2.read ())));
  }

  SCA_CTOR (integrator_sd) {}
};
#endif
```

2nd order Sigma-Delta « SystemC-AMS » decimator

```
#ifndef SAMPLER_SD_H
#define SAMPLER_SD_H

SCA_SDF_MODULE (sampler_sd)
{
  sca_sdf_in < double >in;
  sca_sdf_out < double >out;

  double input;

  void init (int r) {
    in.set_rate(r);
  }

  void sig_proc () {
    out.write(in.read(0));
  }
  SCA_CTOR (sampler_sd) {}
};
#endif
```



SystemC-AMS Decimator 85.3 MHz to 8.53 MHz

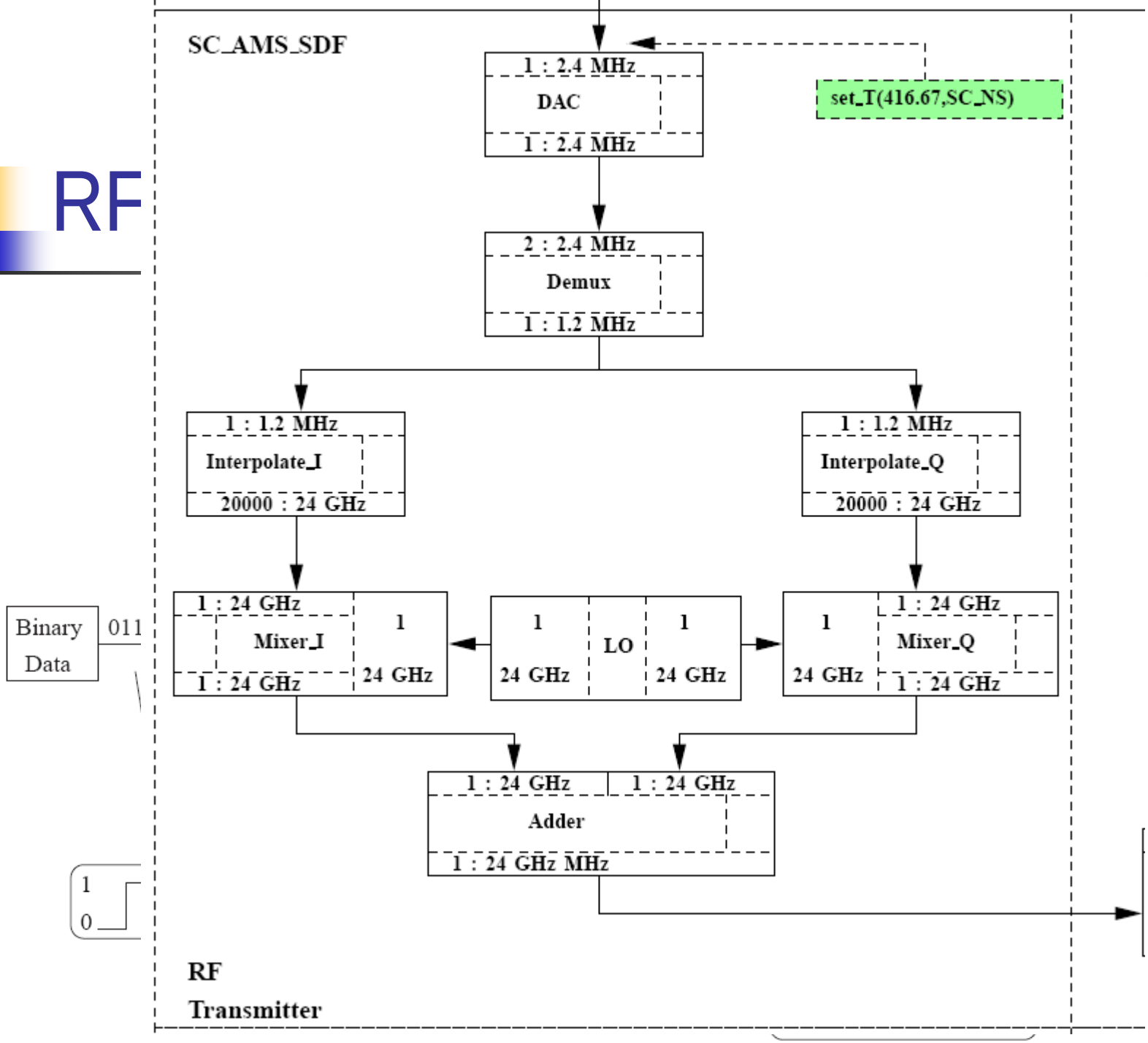
ATMEL Atmega128 BCA Instruction Set Simulator

```
main:
    ldi   R16,0x01
    out   DDRA,R16 ; portA0 output
    ldi   R16,0x00
    out   DDRB,R16 ; portB[7:0] input
loop:
    in  R16,PINB   ; read ADC
    out   PORTA,R16 ; shift Bit 0
    lsr r16
    out   PORTA,R16 ; shift Bit 1
    lsr r16
    out   PORTA,R16 ; ...
    lsr r16
    out   PORTA,R16
    lsr r16
    out   PORTA,R16
    lsr r16
    out   PORTA,R16
    lsr r16
    out   PORTA,R16
    lsr r16
    out   PORTA,R16
    rjmp  loop      ; and loop
```

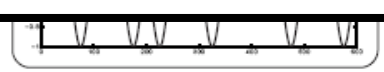
HEX File

```
:100000000C9446000C9463000C9463000C94630001
:100010000C9463000C9463000C9463000C946300D4
:100020000C9463000C9463000C9463000C946300C4
:100030000C9463000C9463000C9463000C946300B4
:100040000C9463000C9463000C9463000C946300A4
:100050000C9463000C9463000C9463000C94630094
:100060000C9463000C9463000C9463000C94630084
:100070000C9463000C9463000C9463000C94630074
:100080000C9463000C9463000C94630011241FBE55
:10009000CFEFD0E1DEBFCDBF11E0A0E0B1E0ECEEEC
:1000A000F0E000E00BBF02C007900D92A030B10756
:1000B000D9F711E0A0E0B1E001C01D92A030B10776
:1000C000E1F70C9465000C940000CFEFD0E1DEBFA7
:1000D000CDBF8FEF80933A00109237008FEF80935F
:0C00E0003B00809136008093 3B00F8CF7D
:00000001FF
```

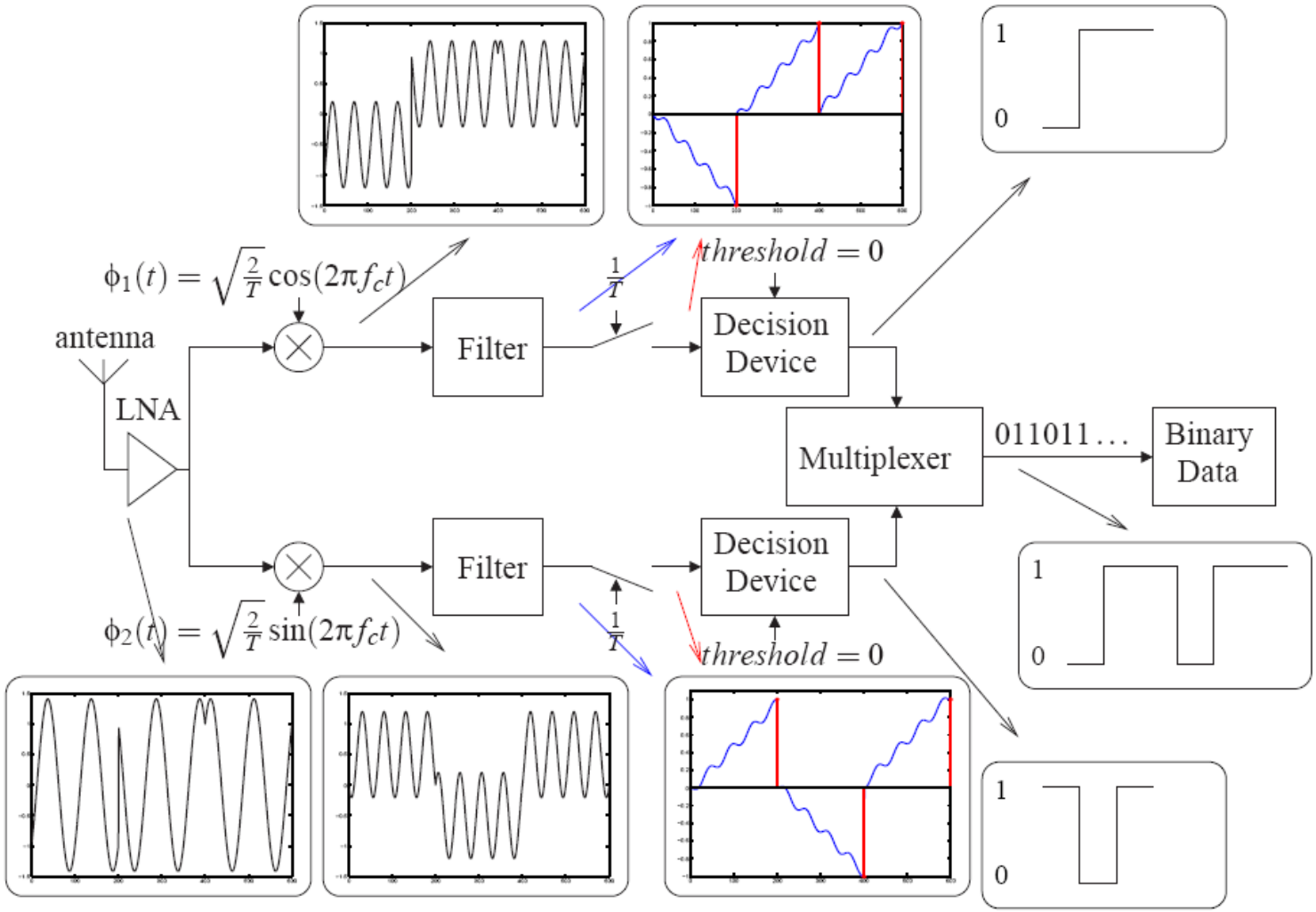
RF


SC_AMS_SDF
1 : 2.4 MHz
DAC
1 : 2.4 MHz
set_T(416.67,SC_NS)
2 : 2.4 MHz
Demux
1 : 1.2 MHz
1 : 1.2 MHz
Interpolate_I
20000 : 24 GHz
1 : 1.2 MHz
Interpolate_Q
20000 : 24 GHz
Binary
Data
011
1 : 24 GHz
Mixer_I
1 : 24 GHz
1
24 GHz
1
24 GHz
LO
1
24 GHz
1 : 24 GHz
Mixer_Q
1 : 24 GHz
1
24 GHz
1 : 24 GHz
Adder
1 : 24 GHz MHz
1 : 24 GHz
1
0
RF
Transmitter

```
SCA_SDF_MODULE (mixer)
{
  sca_sdf_in < double >in;
  sca_sdf_out < double >out;

  double ampl; bool func; double wc; double Ts;

  double dc_offset; double phase_mismatch; double gain_mismatch; double pulsation_offset;

  …

  void sig_proc () {
    double t = in.get_time().to_seconds()+Ts;   // get_time() returns previous time
    switch(func){
      case COS:
        out.write((in.read()*cos((wc+pulsation_offset)*t+
           phase_mismatch/2)*ampl+dc_offset)*(1+(gain_mismatch/2)));
        break;
      case SIN:
        out.write((in.read()*sin((wc+pulsation_offset)*t-
           phase_mismatch/2)*ampl+dc_offset)*(1-(gain_mismatch/2)));
        break;
    }
  }
  void init(bool f, sc_time tc, sc_time tb, sc_time ts,double dc_o, double g_m,
     double ph_m, double f_o){
  …
  }
  SCA_CTOR (mixer) {}

};
```



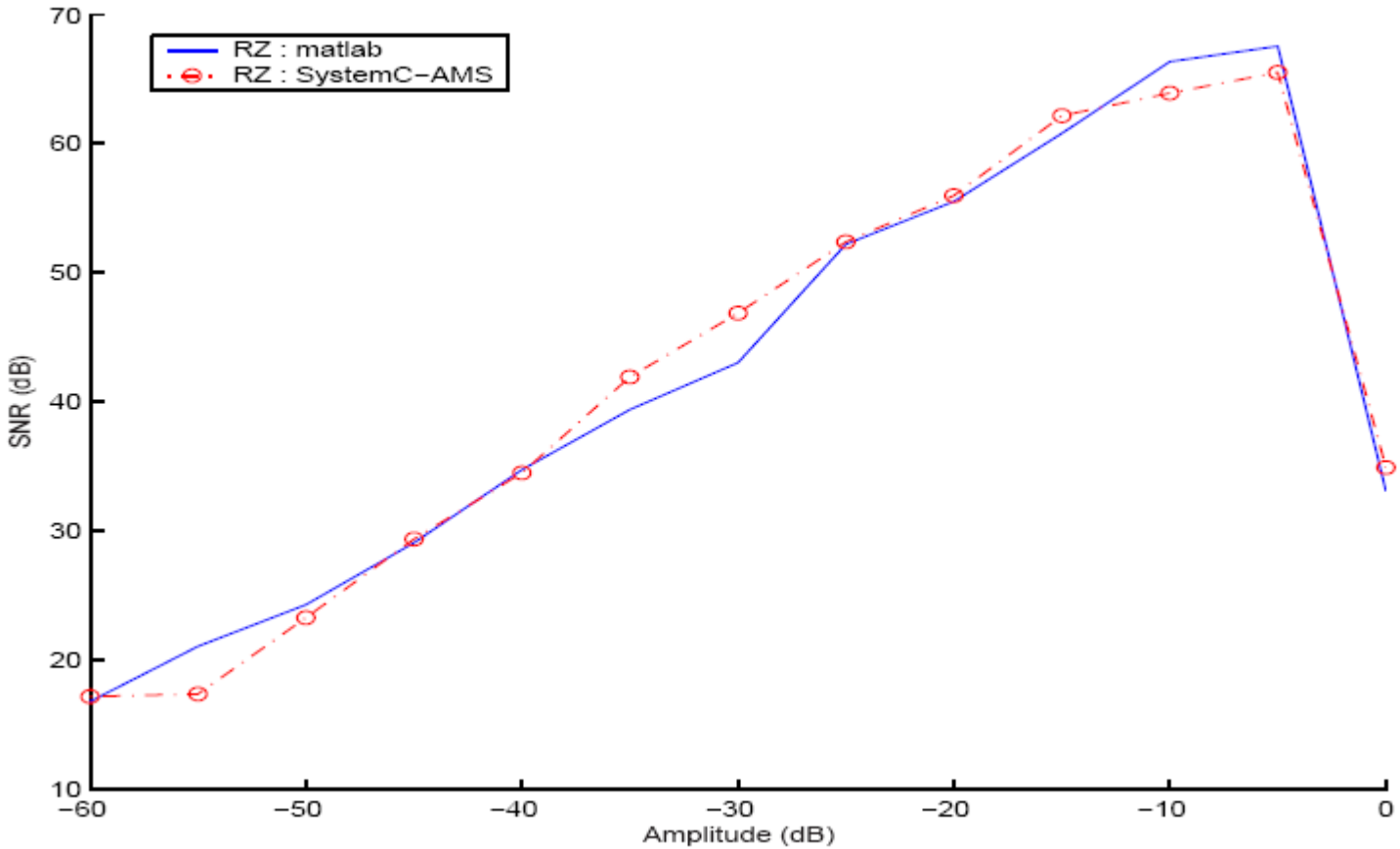
RF Transceiver: Receiver



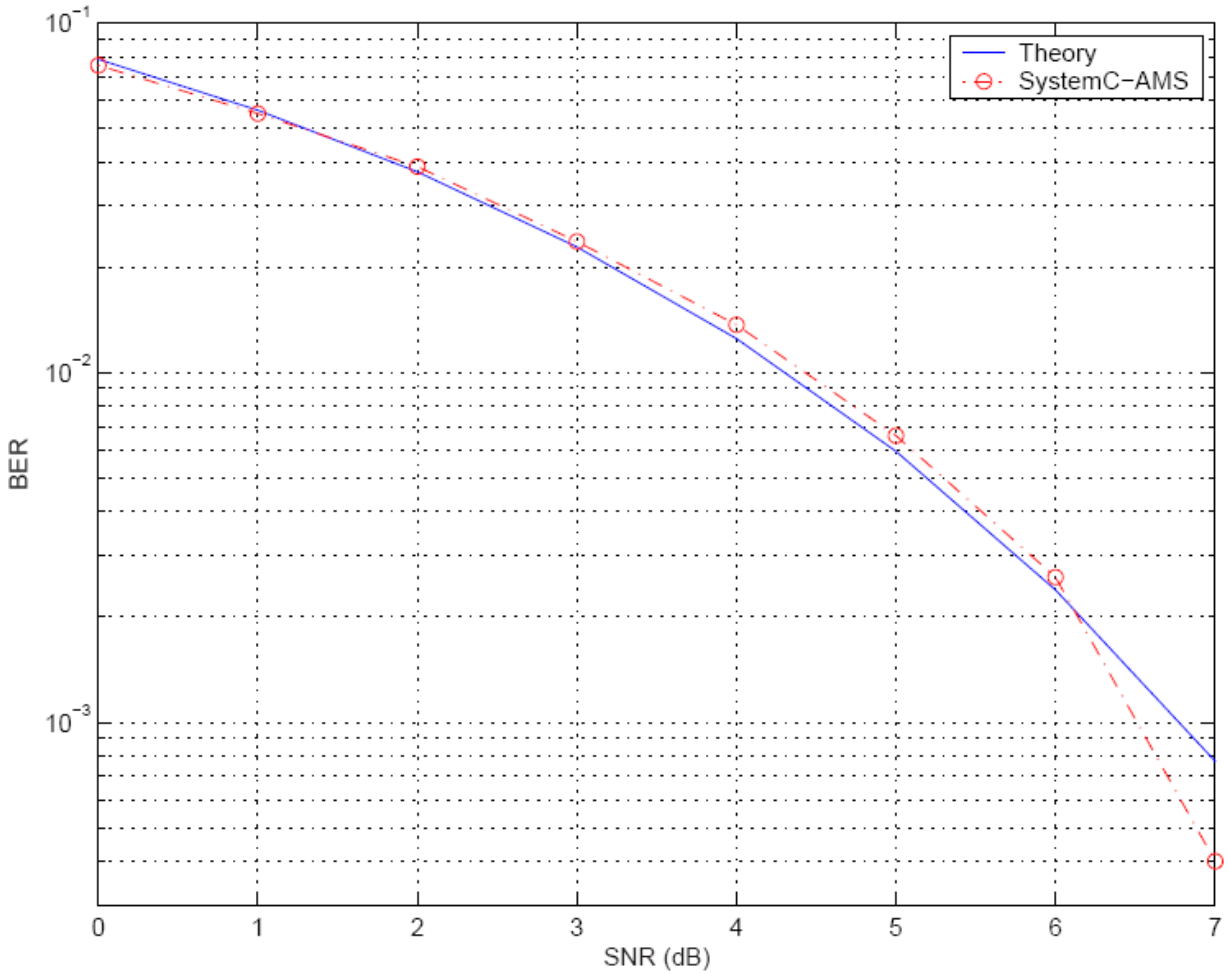
Simulation results

	Configuration	Simulation	Matlab	SystemC-AMS
ADC	OSR=64 8 bits output	1 ms 16*1024 pts for output	1.605s	0.934s
RF	2.4 GHz carrier freq.	416.67 μs 10^3 pts for digital part, 10^7 pts for RF i/o	2m30.746s	54.360s
2-mote WSN	Same settings	416.67 μs	–	3m1.654s

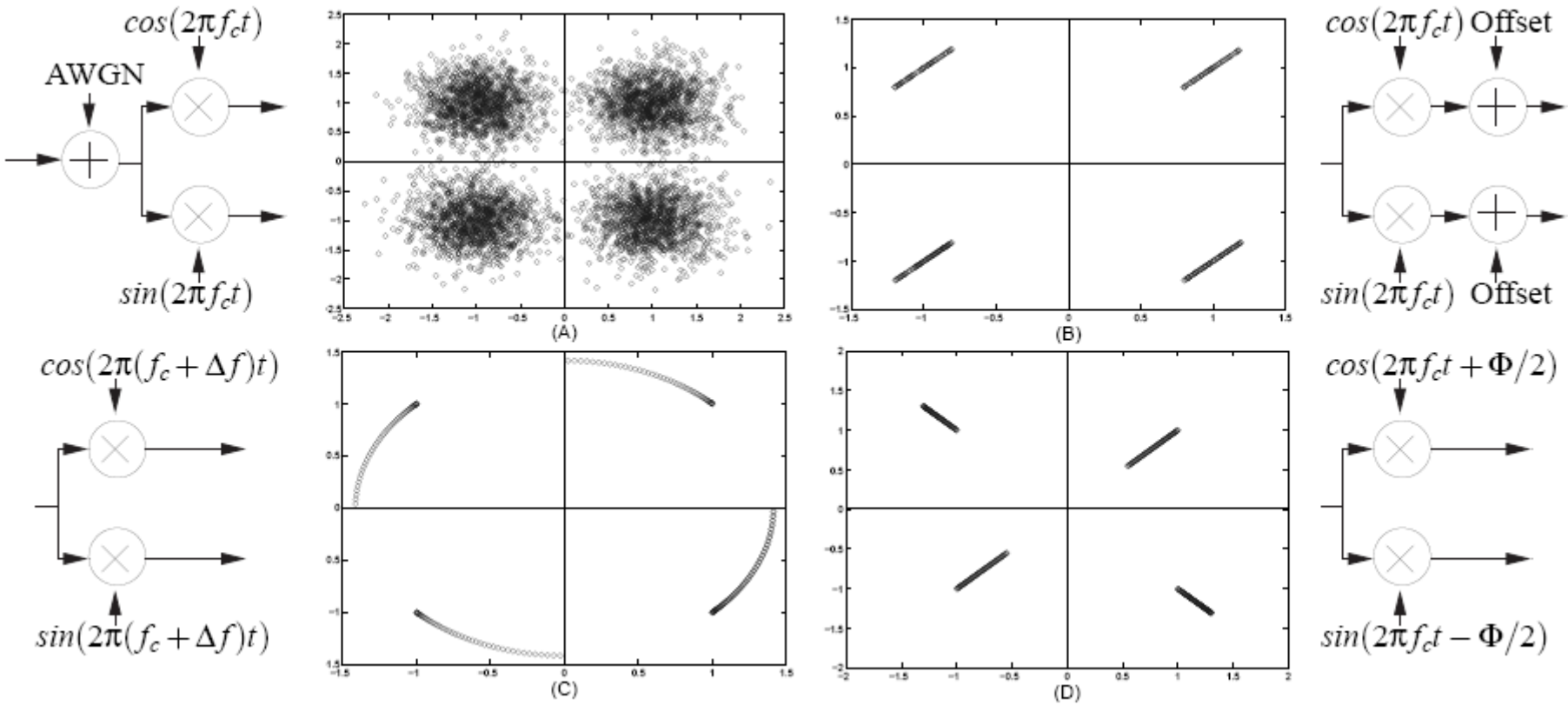
Accuracy SystemC-AMS vs Matlab



Bit Error Rate (BER)



Taking non-idealities into account



Ongoing research

- Non-linearities, LNA, Noise, CP1, IIP3
- Baseband-equivalent

Simulation	SC-AMS classical simulation with refinments	SC-AMS BB equivalent RF simulation
1000 bits transmission	1m2.958s	0m0.036s
DC offset -1e5:5e3:1e5	0m19.916s	0m0.018s
Freq. offset 0:20:1e3	0m24.918s	0m0.022s
Phase mismatch $0:\frac{\pi}{360}:\frac{\pi}{4}$	0m44.407s	0m0.031s