SystemC and Mixed-Signal Simulation Concepts

Karsten Einwich,
Christoph Clauss,
Peter Schwarz

Fraunhofer IIS Design Automation Dept.
Dresden

4. European SystemC Users
Group Meeting
Copenhagen 2001

SystemC and Mixed-Signal Simulation Concepts

SystemC - models in standard M/S-simulators

SystemC extensions for telecommunication

RF-extensions

SystemC extension for automotive

Conclusions / Observations

Motivation

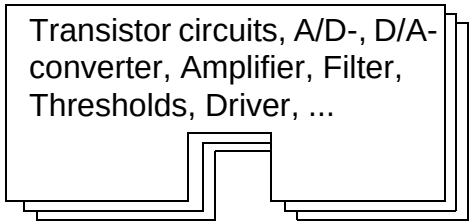
User models

Embedding in tools on different platforms

Embedding into different tools

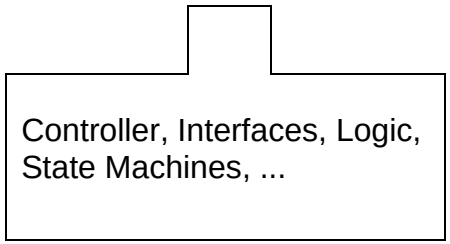
IP-protection

Analog



Saber, AdvanceMS, Matlab/Simulink, ...

Digital



SystemC

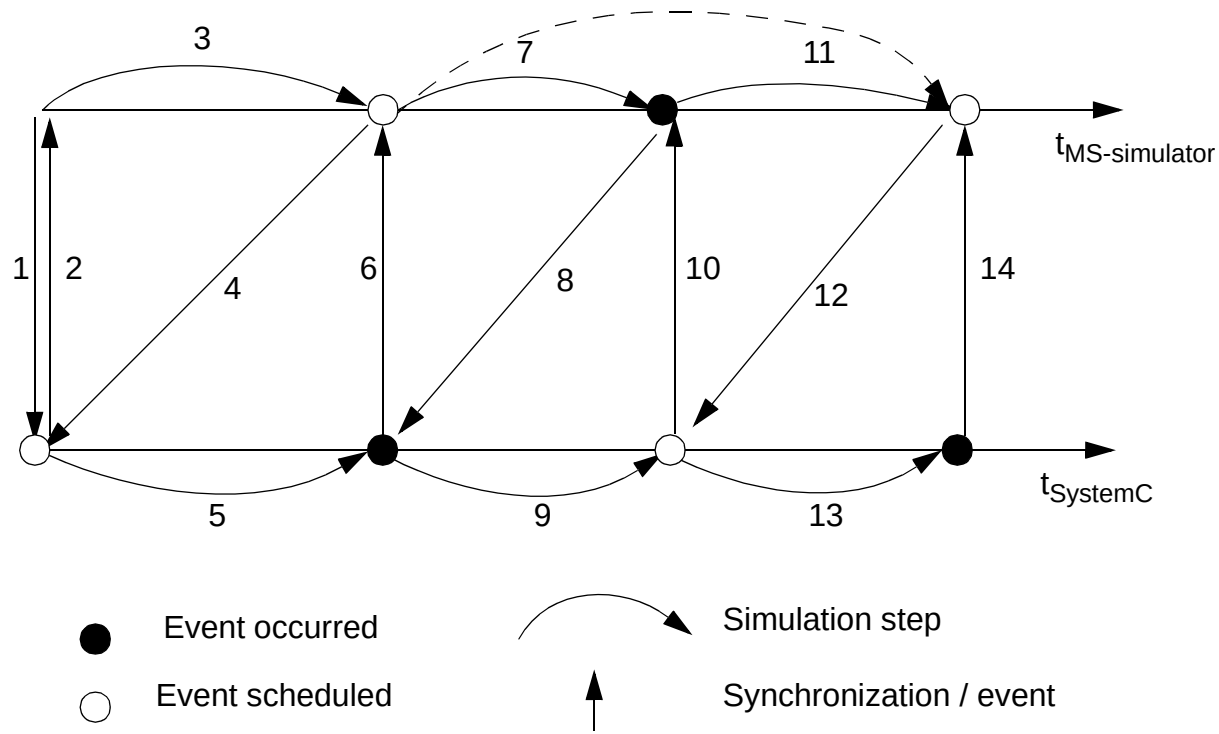
Synchronization principle

Event driven

MS-simulator runs ahead

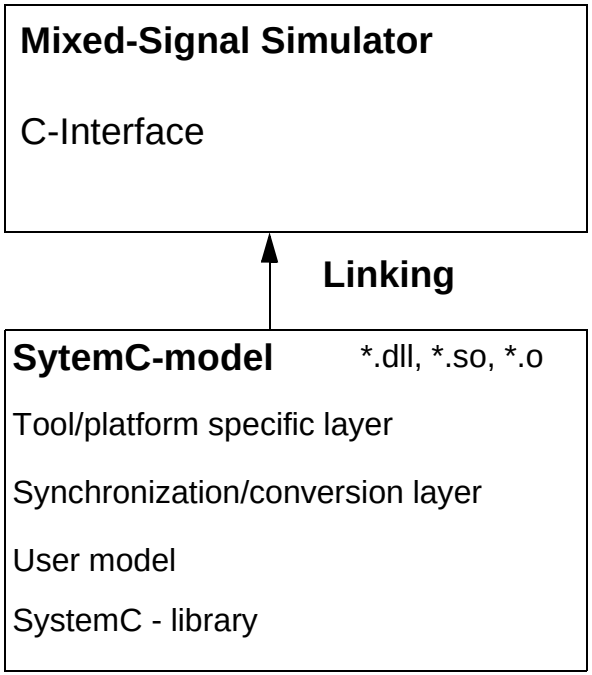
Requirements MS-simulator:
- C-interface
- Foreign block must be scheduleable by time and event

Restriction:
- No zero-delay (delta cycle) iteration possible



Implementation

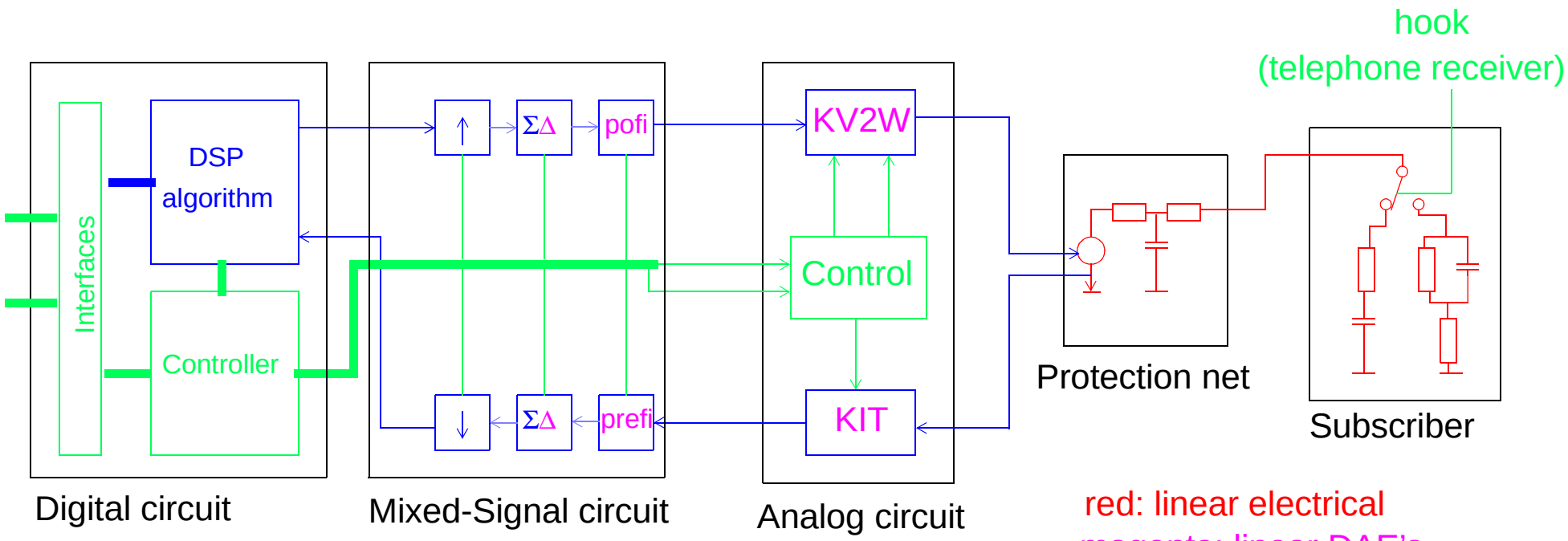
Principle:



Problems:

- SystemC next event information
- Dynamic simcontext creation/destroy
- Signal update
- Entity/Type exploration

Motivation Mixed-Signal extensions



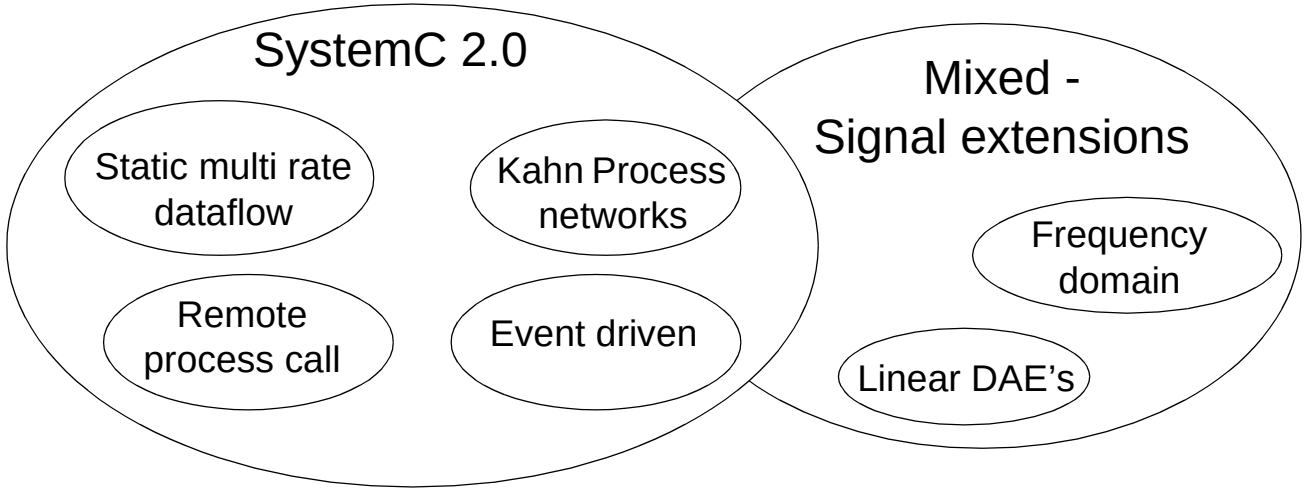
red: linear electrical
magenta: linear DAE's
blue: dataflow
green: digital (SystemC)

Mixed-Signal extensions

Including via Dataflow

Using optimized dataflow scheduler for static cluster

Simple and effective synchronization mechanism



Modeling capabilities of a linear solver

Linear behavior models:
- Transfer function (Hp, Lp, ...)
- State space systems
- Pol-Zero

$$A\dot{x} + Bx + q(t) = 0$$

Linear networks
- R,L,C,V,I, ...
- Linear transformer models
- Line models, ...

Solvers:
- Euler backward
- Trapezoidal

Frequency domain simulation

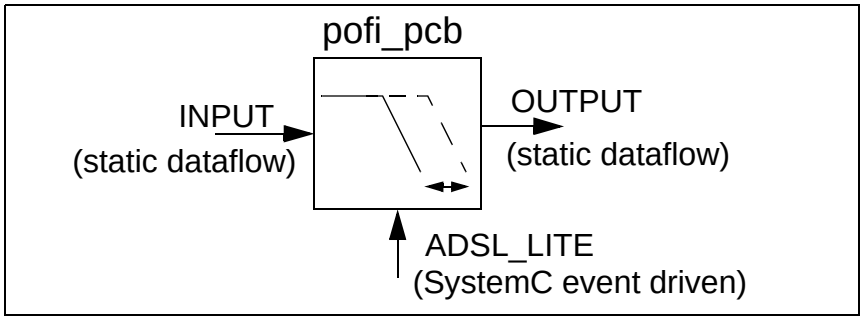
Mixed-Signal-Systems specified in frequency domain

Networks / linear differential equation	Solving equation system in frequency domain
Dataflow blocks	1. Complex arithmetic 2. Optional frequency domain implementation
Digital filter (SystemC event driven)	Optional frequency domain implementation

Dataflow block with SystemC in-port

```
SDF_MODULE(pofi_pcb)
{
 sdf_inport     <double>  INPUT;          //dataflow inport
 sc2sdf_inport<bool>      ADSL_LITE;      //SystemC inport
 sdf_outport   <double>   OUTPUT;         //dataflow outport

 double FG0, FG1, K, h;                   //parameters

 LTF_ID ltf_id0, ltf_id1;
 vector<double> A0,A1, B0,B1, S;

 void attributes() {              //port attributes for synchronization
        ADSL_LITE.h=h;            //h sample time
        }


 void init() {
        double wpre0;                    double wpre1;
        wpre0=2.0*M_PI*FG0;              wpre1=2.0*M_PI*FG1;
        A0(0)=1.0;                       A1(0)=1.0;
        A0(1)=1.41/wpre0;                A1(1)=1.41/wpre1;
        A0(2)=1.0/wpre0/wpre0;           A1(2)=1.0/wpre1/wpre1;
        B0(0)=K;                         B1(0)=K;
}
```

```
void sig_proc() {
        if(ADSL_LITE)
                OUTPUT=LTF(A1,B1,S,ltf_id1,INPUT);
        else
                 OUTPUT=LTF(A0,B0,S,ltf_id0,INPUT);
        }

        SDF_CTOR(pofi_pcb);
};
```

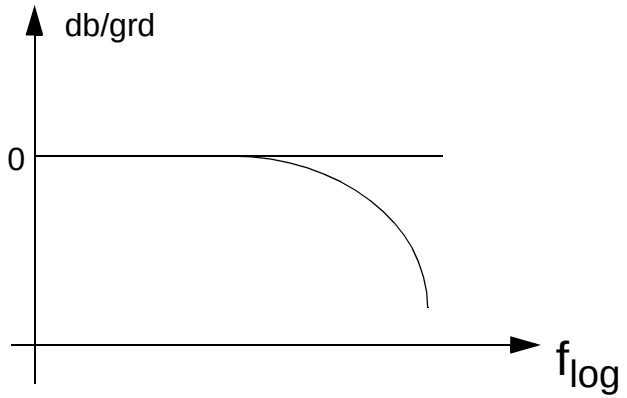


$$H(s) = \frac{K}{1 + \frac{1,41}{(2\pi FG)^2}s^2 + \frac{1}{2\pi FG}s}$$

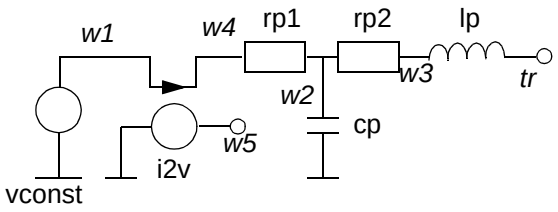
Frequency domain implementation

```
SDF_MODULE(delay)
{
    sdf_inport<double>  inp;
    sdf_outport<double> outp;

    sdf_para<unsigned long>  delays;
    sdf_para<double>         init_val;

    void defaults() {
                        delays  =1;
                        init_val=0.0;
                      }


    void attributes() {
                        outp.delay=delays;
                      }



  void init() {
          for(unsigned long i=0;i<delays;i++)  outp[i]=init_val;
            }
```

```
    void sig_proc() {
                        outp=inp;
                      }

    void ac_domain() {
                        outp.freq.ampl=inp.freq();
                        outp.freq.deg  =delays*inp.T() * 360 * freq;
                      }

    SDF_CTOR(delay);
};
```



Netlist example




```
                :
elec_wire   w1, w2, w3, w4, w5, tr;
elec_gnd    gnd;        //reference node

    double Rp1=60.0, Rp2=40.0;
    double Cp=1e-12, Lp=1e-3;

    V vconst(w1,gnd,2.0);

    R rp1 (w4,w2,Rp1);
    R rp2 (w2,w3,Rp2);
    C cp  (w2,gnd,Cp);
    L lp   (w3,tr,Lp);
    CCVS i2v (w5,gnd,w1,w4,1.0);
    //Current Controlled Voltage Source
                :
```

```
// signal tracing
trace tr1(MATLAB,“tr1.dat“);

tr1.add(&w1);   //node voltage
tr1.add(&lp);    //current through lp
```

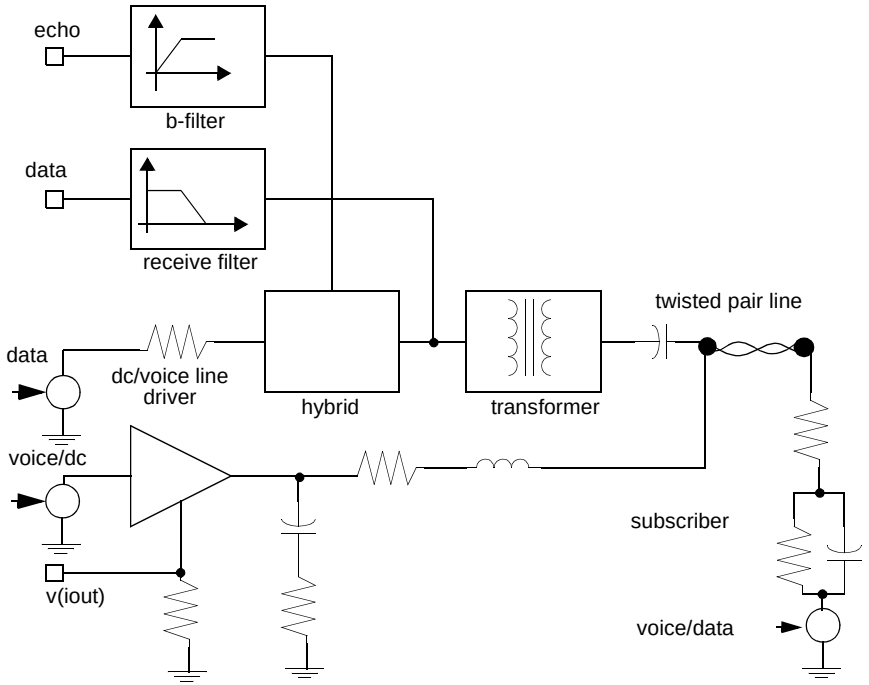
Results

ADSL-Line driver front-end

Ca. 50 linear elements

Different lines via coefficient file

Time step size 1.0/17.664MHz



Performance estimation

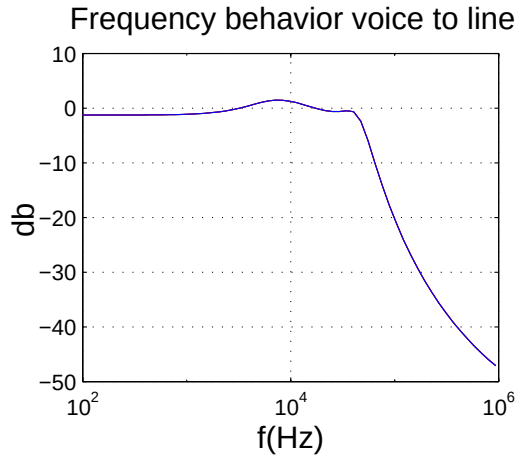
Input: Multi tone signal from 100Hz to 1MHz on voice
Samplerate: 1.0/17.664MHz
Time interval: 40 ms

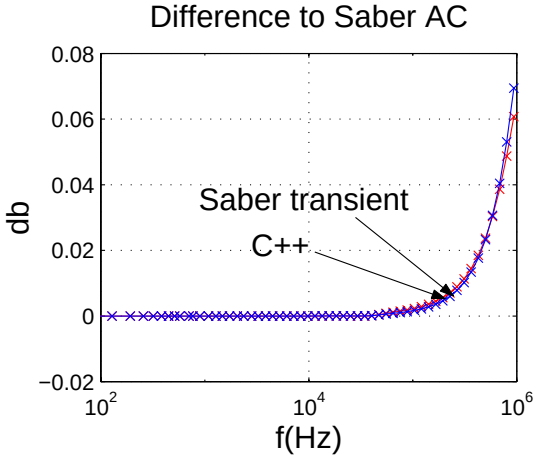
Graphs:
Amplitude and phase over frequency (for transient fft-result)

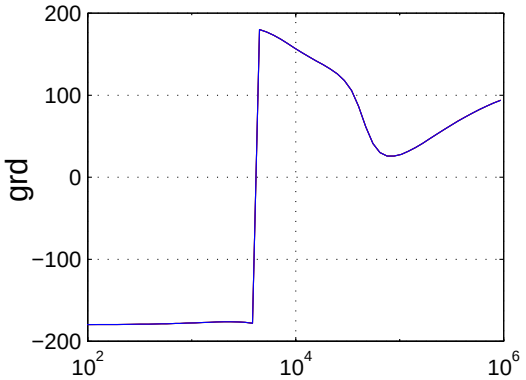
Comparison between:
- Saber - AC- Analysis (reference)
- Saber - transient analysis
- C++ (MixSigC) transient analysis

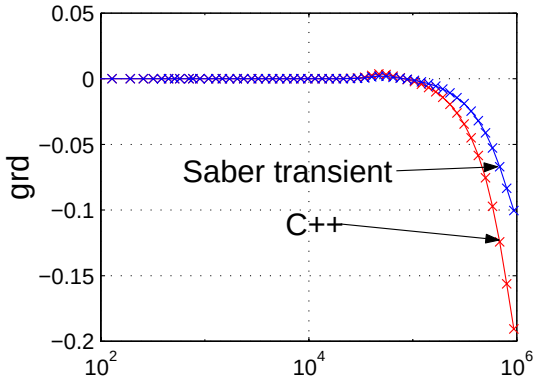
Computation time:
Saber transient (default parameters): 2670 sec.

C++ transient: 122 sec.







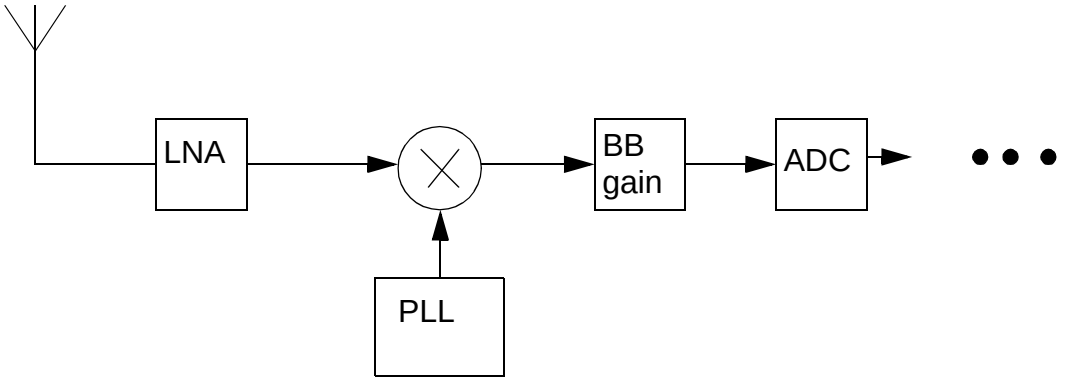


RF-Baseband

Modeling using dataflow scheduler

e.g. Literature:

G. Vandersteen et. al
„A methodology for efficient high-level dataflow simulation of mixed-signal front-ends of digital telecom transceivers“

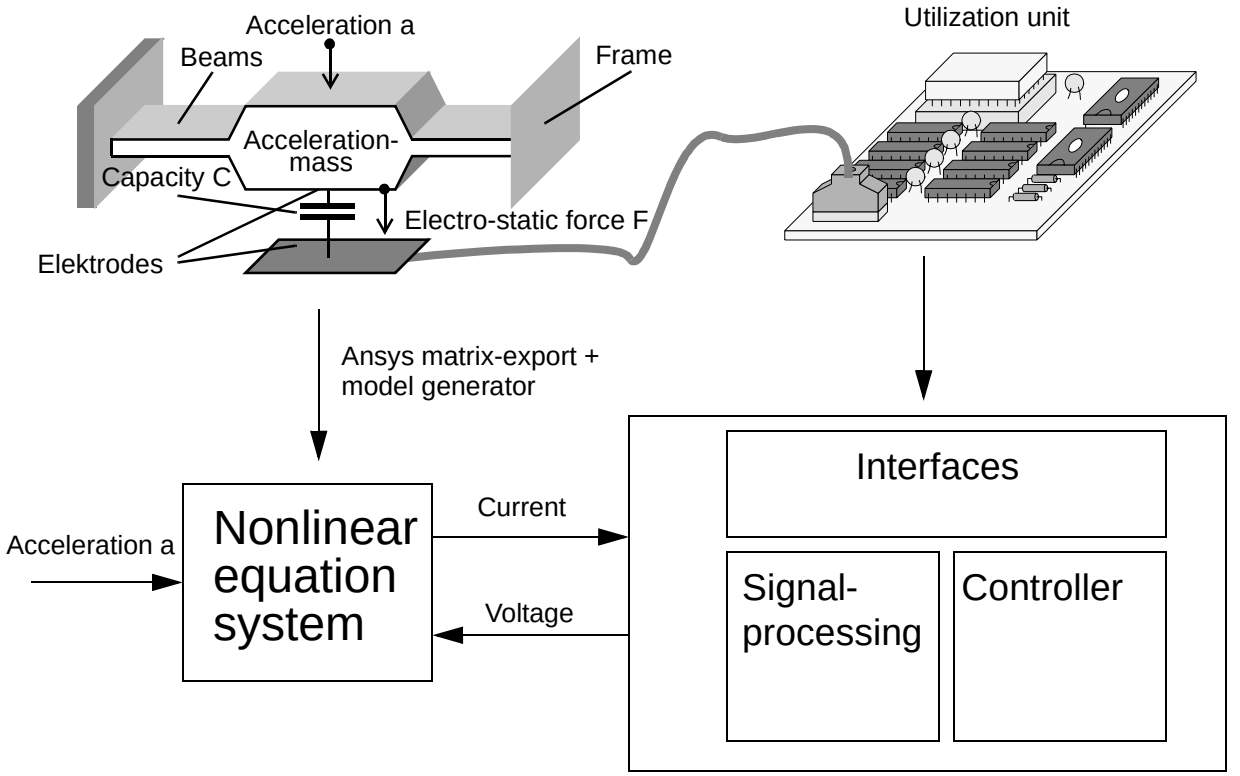


Nonlinear equation in dataflow blocks

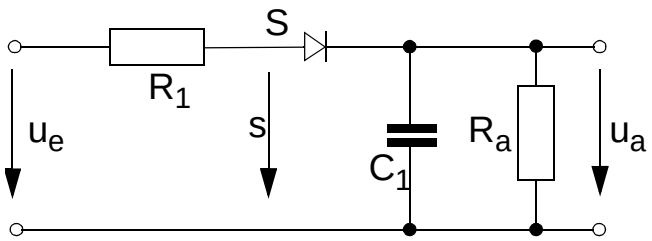
Embedded nonlinear equation solver

Dynamic time step control internally

Externally dataflow time steps



Example for nonlinear equation system



Network equations:

$0 = C_1 \dot{u}_a + u_a/R_a - I_s\,(\exp((s - u_a)/(m_d\, U_t)) - 1)$

$0 = (s - u_e)/R_1 + I_s\,(\exp((s - u_a)/(m_d\, U_t)) - 1)$

```
SDF_MODULE(nonlinear_example)
{
    sdf_inport<double>    ue_sdf;
    sdf_outport<double> ua_sdf;

                          :
   out_eq ua;   in_eq   ue; state_eq s;

    void init() {
                    ua=0.0;
                    s=0.0 }

    void equations()  {
                        null[0] = C1*d_t(ua) + ua/Ra - Is*(exp((s-ua)/md/Ut) - 1);
                        null[1] = (s-ue)/R1 + Is*(exp((s-ua)/md/Ut) - 1); }
    void sig_proc() {
                         ue=ue_sdf;
                         solve_equations();
                         ua_sdf=ua; }
                            :
```

Conclusions/Observations

Different Models of Computation useful for analog domains also

Excellent performance can be reached by combination of MoC's

On system level the dataflow MoC's is well adapted for analog MoC integration

Conceptual there are no restrictions for analog integration (the question is what makes sense)