RINCON™ - A Rigorous Tool for Accurate RF/Microwave Modeling and Simulation

D.Goodman <doug@ridgetop-group.com>,
G. Serdyuk <gserdyuk@ridgetop-group.com>
Ridgetop Group, Inc.
3580 West Ina Road
Tucson, AZ 85741

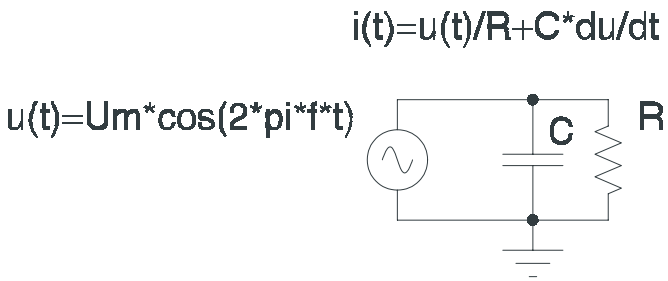
1 Introduction

A key benefit of creating a new analysis tool is the ability to incorporate recent advances in modeling technology to provide the RF/Microwave Designer with enhanced capabilities. Using VHDL-AMS as a base, the Authors have developed a powerful Modeling/Simulation tool with unparalleled modeling capabilities for rigorous and accurate modeling of complex devices.

Conventional netlist-oriented primitive models do not provide the fidelity necessary nor do they provide the interchangeability that support collaborative design activities. The Authors have previously described the VHDL-FD extension to the VHDL-AMS language [1] that incorporates frequency domain capability for rigorous RF and Microwave device modeling. This paper describes examples of applying the VHDL-FD toward the creation of new models.

2 Simple VHDL-FD example

To present features of VHDL-FD, let us consider simple circuit:



This circuit may be represented by simple netlist in terms of VHDL-FD:

```
entity simple_circuit is  -- declare of some circuit
end entity simple_circuit
--
architecture netlist of simple_circuit is -- define circuit
 constant Um: real := 1;     -- amplitude value
 constant R: real := 100;    -- resistance
 constant C: real := 5.e-12; -- cap 5pF
 constant fr: real := 1.e+9; -- frequency 1GHz
 terminal a: electric;       -- single node
begin
  R1: entity res(eq) generic map(R) port map (a, GROUND);
  C1: entity cap(eq) generic map (C) port map (a, GROUND);
  E1: entity source(frequency_domain)
        generic map( Um, frequency) port map (a, GROUND);
end architecture;
```

This netlist is similar to SPICE netlist (in general) and defines some interconnection of elements. Elements are referenced by their entity name and architecture name - like "res(eq)" where "res" is entity name and "eq" - architecture name. The language allows predefinition of some objects in upper section (like "constant" and "terminal").

But this circuit description is not complete, because it knows nothing about what does it mean "res(eq)" or "cap(eq)". To give the simulator an idea about properties of component it is necessary to define it. It may be done in term of another components (like SPICE's subcircuits), but may be done in terms of equations, which is distinct feature of language, which is derived from VHDL-AMS [2]:


```
entity res is
  generic (r: real); -- generic parameter
  port (terminal a, b: electrical); -- ports to be connected
end entity res;
architecture eq of res is
quantity ur across ir through a to b; -- define quantities
begin
    ir==ur/r; -- quantities interrelation - it is EQUATION
end architecture;
```

This code contains a few new statements:

* definition of quantities "ur" and "ir" as through and across quantities in branch a-b;

* simultaneous statement "==", which represents equation in the module.

Now it is known, what is it "res" and what is the architecture "eq" of it. A similar way is used to define other components of the circuit.

But this is not the only way to describe abovementioned circuit. As you can see, VHDL-FD allows the user to handle with equations, not only with components and their interconnections. So we can define another architecture of abovementioned entity, which describes the circuit:

```
architecture equations of simple_circuit is -- define circuit
 constant Um: real := 1;     -- amplitude value
 constant R: real := 100;    -- resistance
 constant C: real := 5.e-12; -- cap 5pF
 constant fr: real := 1.e+9; -- frequency 1GHz
 quantity Va: real;
 quantity i, ic: real;
begin
 i== Va/R+ic; -- source current = resistor current + cap curent
 ic'FD==Va'FD*(math_j*2*math_pi*FREQUENCY())*C; -- cap. current
 if (FREQUENCY() = fr) use -- define source voltage
   Va'FD == Um/sqrt(2.);
 else use
   Va == ZERO; -- it is predefined: ZERO = COMPLEX(0,0);
 end use;
end architecture;
```

Here you can see a few new constructions:
* free quantity declaration "quantity";
* frequency-domain attribute " 'FD ";
* if-use-else construction;
* predefined function returns current frequency "FREQUENCY()".

This architecture defines behavior of the circuit not using terminals - only unknown variables "quantities". These two approaches to define circuit and/or system can be mixed to achieve optimum trade-off between degree of detail and generalization of different parts of (sub-) system.

3 Modeling using VHDL-FD

Proposed VHDL-FD language variant is capable to express even most complex behavior both in time- and frequency- domain. To illustrate modeling capabilities, we describe modeling process for different components. Now consider more rigorously the following examples.

3.1 Lumped passive components

To describe usual circuit component it is necessary to define topology (i.e. branches and associated quantities) and relations between branch quantities. See the following example for a resistor.

Component description is divided onto two parts - interface ("entity") and body itself ("architecture").

```
entity res is                        -- entity "res"
  generic (r: real);                 -- generic parameter
  port (terminal a, b: electrical);  -- ports of entity
```

```
end entity res;
--
architecture eq of res is          -- define architecture
quantity ur across ir through a to b; -- quantities
begin
    ir==ur/r;       -- quantities interrelation
end architecture;
```

The architectures provide flexibility in avoiding problems found in conventional simulators, such as divide by zero situations, e. g. as above in case of r=0. This is handled by including a test for a zero value in the denominator, then calculating the value, or assigning a value to current reflecting a zero value of resistance:

```
architecture eq_zero of res is
  quantity ur across ir through a to b;
begin
  if( r /= 0) use
    ir==ur/r;   -- usual interrelations between current and voltage
  else
    ur==0;      -- if resistance == 0 => voltage == 0 too
  end use;
end architecture;
```

Both forms of architecture can be used in circuit-level description:

```
entity gilbert is
end entity gilbert;
--
architecture component of gilbert is
...
R7: entity res(eq)      -- for this element resistance != 0
  generic map (1500) port map (node1, node7);
R8: entity res(eq_zero) -- and for this element may be == 0
  generic map (0) port map (node6, node7);
...
```

Similar code describes other simplest passive components:

```
-- CAPACITOR -------------------------------------
entity cap is
  generic (c: real);
  port (terminal a, b: electrical);
end entity cap;
--
architecture eq of cap is
  quantity uc across ic through a to b;
begin
  ic'FD==2*math_pi*math_j*FREQUENCY()*c*uc'FD;
  -- FD extension allows to operate in frequency domain: Ic == j*2*PI*f*Uc
end architecture;
--
-- INDUCTOR --------------------------------------
entity ind is
  generic (l: real);
  port (terminal a, b: electrical);
end entity ind;
--
architecture eq of ind is
  quantity u across i through a to b;
begin
  if (FREQUENCY() /= 0) use
    i'FD==u'FD/(2*math_pi*math_j*FREQUENCY()*l);
    -- usual  I=U/ZL
  else
    u'FD==0; -- if Zl ==0; -> UL == 0 too
  end use;
end architecture;
```

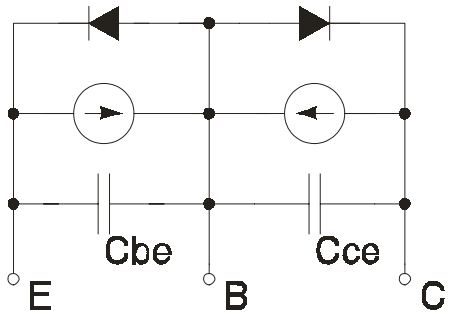

It is possible to define more complex behavior for the component, e.g. introduce the temperature dependency of the component. Sure, it impacts interface too:

```
-- CAPACITOR with temperature dependency ------------------
entity cap_temp is
  generic (c: real, ct: real);
  port (terminal a, b: electrical;
        quantity temp: thermo );   -- input parameter to pass temperature
into model
end entity cap_temp;
--
architecture eq of cap_temp is
  quantity uc across ic through a to b;
begin
  ic'FD==2*math_pi*math_j*FREQUENCY()*uc'FD*(c+ct*temp);
  -- capacitance is temperature-dependent
end architecture;
```

The foregoing examples describe VHDL-FD capabilities in a very basic form, and illustrate the capabilities it provides.

3.2 BJT

To present more modeling capabilities of VHDL-FD, consider the bipolar junction transistor model.



In this example no parameters will be defined in the interface, but any parameters can be included at a later time. And (in first model) no capacitances will be included.

```
entity bjt is
  port (terminal e,b,c : electric);
end entity;
--
-- EBERS MOLL BIPOLAR SIMPLEST ---------------------
architecture em1 of bjt is
  constant i0: real := 1.e-12;  -- saturation current of diode
  constant alpha: real := 35;   -- exponent factor
  constant af: real := 0.99;    -- forward current gain
  constant ar: real := 0.99;    -- reverse current gain
  quantity Vbe across Ibe through b to e; -- quantities of base-emmiter junction
  quantity Vbc across Ibc through b to c; -- quantities of base-collector
junction
begin
Ibe==i0*(exp(alpha*Vbe)-1) - ar*i0*(exp(alpha*Vbc)-1);
  -- base-emmiter current versus Vbe and Vbc; exp is defined outside
Ibc==i0*(exp(alpha*Vbc)-1) - af*i0*(exp(alpha*Vbe)-1);
  -- base-collector current
end architecture;
```

This architecture defines only static behavior and do not include capacitances. To include capacitances the following line are inserted into model:

```
architecture em_cap of bjt is
  constant i0: real := 1.e-12;  -- saturation current of diode
  constant alpha: real := 35;   -- exponent factor
  constant af: real := 0.99;    -- forward current gain
  constant ar: real := 0.99;    -- reverse current gain
  constant Cbe: real := 10.e-12;-- Cbe := 10pF
  constant Cbc: real := 7.e-12; -- Cbc := 10pF
  quantity Vbe across Ibe through b to e; -- quantities of base-emmiter junction
  quantity Vbc across Ibc through b to c; -- quantities of base-col junction
begin
  Ibe==i0*(exp(alpha*Vbe)-1) - ar*i0*(exp(alpha*Vbc)-1);
    -- base-emmiter current versus Vbe and Vbc; exp is defined outside
  Ibc==i0*(exp(alpha*Vbc)-1) - af*i0*(exp(alpha*Vbe)-1);
    -- base-collector current
  -- these two constant capacitances are defined earlier
  -- - in section "Lumped passive components"
  CBE: entity cap(eq) generic map (Cbe) port map (b, e);
  CBC: entity cap(eq) generic map (Cbc) port map (b, c);
end architecture;
```

In this case we have added separate components to the circuit: CBE, CBC. But, sure, it was possible to define currents of branches via equations at this level:

```
...
quantity ic_bc, ic_be: electric;
begin
  Ibe == i0*(exp(alpha*Vbe)-1) -ar*i0*(exp(alpha*Vbc)-1) +ic_be;
  Ibc == i0*(exp(alpha*Vbc)-1) -af*i0*(exp(alpha*Vbe)-1) +ic_bc;
  ic_be'FD == Cbe*Vbe'FD* (math_j*2*math_pi*FREQUENCY());
  ic_bc'FD == Cbc*Vbc'FD* (math_j*2*math_pi*FREQUENCY());
end architecture;
```

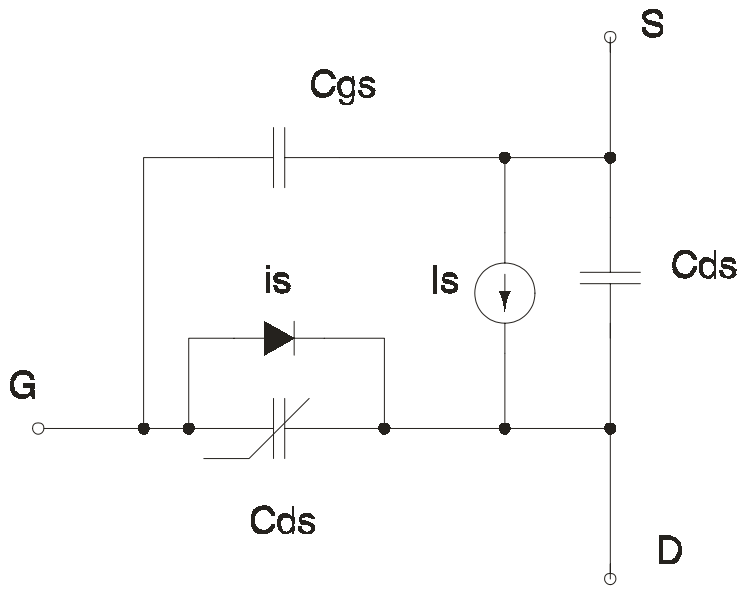
Let us define BJT model with nonlinear capacitances. In this case it is better to operate with charges:

```
constant Vt: real := 0.02585;  -- thermal voltage
constant  tau: real:= 1.e-9; -- transition time
...
quantity Qbc, Qbe: electric;
quantity ic_be, ic_bc: electric;
begin
  Ibe == i0*(exp(alpha*Vbe)-1) -ar*i0*(exp(alpha*Vbc)-1) +ic_be;
  Ibc == i0*(exp(alpha*Vbc)-1) -af*i0*(exp(alpha*Vbe)-1) +ic_bc;
  ic_be'FD == Qbe'FD* (math_j*2*math_pi*FREQUENCY()); -- current via charge
  ic_bc'FD == Qbc'FD* (math_j*2*math_pi*FREQUENCY());
  Qbe == tau*Ibe         -- diffusion component
       - 2*Cbe_junc*Vt**0.5*(Vt-Vbe)**0.5; -- junction component
  Qbe == ...; -- similar
end architecture;
```

Consider, that there are circular dependencies between Ibe and Qbe. It is allowed, so user can define dependencies between variables in very flexible manner.

3.3 MESFET

MESFET model schematics may be considered like the following:



Build VHDL-FD model for it. First, define entity:

```
entity mesfet is
   port (g,s,d: electric);
end entity mesfet;
```

Architecture will be the following:

```
architecture eq of mesfet is
  constant Cds: real:= 5.e-12; -- capacitances
  constant Cdg: real:= 1.e-12;
  constant Cgs0: real:=1.e-12;
  constant Is0: real :=1.e-12; -- Schottky junction saturation current
  constant alpha: real := 35; -- exponent
  constant beta: real :=0.05; -- A/V
  constant lambda: real := 0.002; -- g out
  constant aa: real := 2;     -- for tanh
  constant Vth: real :=-1.0l  -- V threshold
  constant Vt: real := 0.02585;  -- thermal voltage
  quantity Vgs across Igs through g to s;
  quantity Vds across Ids through d to s;
  quantity Vgd across Igd through g to d;
  quantity Qgd;
begin
Ids == Is + Icds;
Igs == Icgs;
Igd == Icgd + is;
Icds'FD == math_j*2*math_pi*FREQUENCY()*Vds'FD*Cds;
Icgs'FD == math_j*2*math_pi*FREQUENCY()*Vgs'FD*Cgs;
Is == beta*(Vgs-Vth)*2**(1-lambda*Vds)*tanh(alpha*Vds);
is == Is0*(exp(alpha*Vgs)-1);
Qgd= - 2*Cbe0*Vt**0.5*(Vt-Vbe)**0.5; -- junction charge
Icgs'FD == Qgd'FD*math_j*2*math_pi*FREQUENCY(); -- and current
```

3.4 System-Level Components

VHDL-FD allows the user to define system-level components, not detailed very much. Here are a few examples of such components with comments.

Signal source:

```
entity source is
```



```
    generic (v, fr: real);  port (quantity a : real);
  end entity source;
  architecture eq of source is
  begin
    if (FREQUENCY() = fr) use
      a'FD==COMPLEX(v,0);  -- signal of value v at frequency fr
    else
      a'FD==ZERO;          -- or 0 if else
    end use;
  end architecture;
```

Amplifier:

```
  entity amp is
    generic (k: real); port (quantity a_in, a_out : real);
  end entity amp;
  architecture eq of amp is
  begin
    a_in'FD*k==a_out'FD;  -- trivial
  end architecture;
```

Linear amplifier may be converted into nonlinear one:

```
  entity nonlinrear_amp is
    generic (k1, k2, k3: real); port (quantity a_in, a_out : real);
  end entity amp;
  architecture eq of amp is
  begin
    a_in*k1+a_in**2*k2+a_in**3*k3==a_out; -- nonlinear effects included
  end architecture;
```

Low-pass filter:

```
  entity lpfilter is
    generic (fr: real); port (quantity a_in, a_out : real);
  end entity;
  architecture eq of lpfilter is
  begin
    if (FREQUENCY() <= fr) use
      a_in'FD==a_out'FD;
    else
      a_out'FD==ZERO;
    end use;
  end architecture;
```

Band-pass filter:

```
  entity bpfilter is
    generic (fl, fh: real); port (quantity a_in, a_out : real);
  end entity;
  architecture eq of bpfilter is
  begin
    if (FREQUENCY() >= fl AND FREQUENCY() <= fh ) use
      a_in'FD==a_out'FD;
    else
      a_out'FD==ZERO;
    end use;
  end architecture;
```

High-pass filter and band-stop filter are similar.

Adder:

```
entity adder is
  port (quantity a, b, o : real);
end entity;
architecture eq of adder is
begin
  o==a+b;
end architecture;
```

Multiplier

```
entity multiplier is
  port (quantity a, b, o : real);
end entity;
-- straightforward description of multiplier
architecture eq of multiplier is
begin
  o==a*b;
end architecture;
-- in most cases real multipliers are closer to the "quadrators"
architecture squared_sum of multiplier is
begin
  o==(a+b)**2;
end architecture;
```

Differentiator:

```
entity differentiator is
  port (quantity i, o : real);
end entity;
architecture eq of differentiator is
begin
  o'FD==i'FD*math_j*2*math_pi*FREQUENCY();
end architecture;
```

4 Rincon Library

4.1 Examination of current library

Ridgetop currently provides a library of pre-tested and calibrated models for use with the Rincon™ Harmonic Balance Simulator. These include the following models: Resistor, capacitor, inductor, controlled sources (VCCS, CCVS, CCCS, VCVS), independent sources (I, V, Power), ideal transmission line. Nonlinear components: Diode, Ebers-Moll BJT, MESFET. System components: source, filters, amplifiers, adder, multiplier and differentiator.

4.2 Future Directions in Model Library Growth

In the future, Ridgetop has identified the need for additional models to be added to the basic library. The top candidates that are being considered for the next release include:

Linear Components:
- Temperature dependent components;
- Electromagnetic components;
- Distributed linear components, including transmission lines (strip, microstrip, slot), microstrip stubs, and table-defined n-poles.

Non-Linear Components:
- HEMT;
- p-i-n diode;
- Transformers;
- Also precisely characterized manufacturer's components that are built on the base BJT, MESFET, HEMT and p-i-n models.

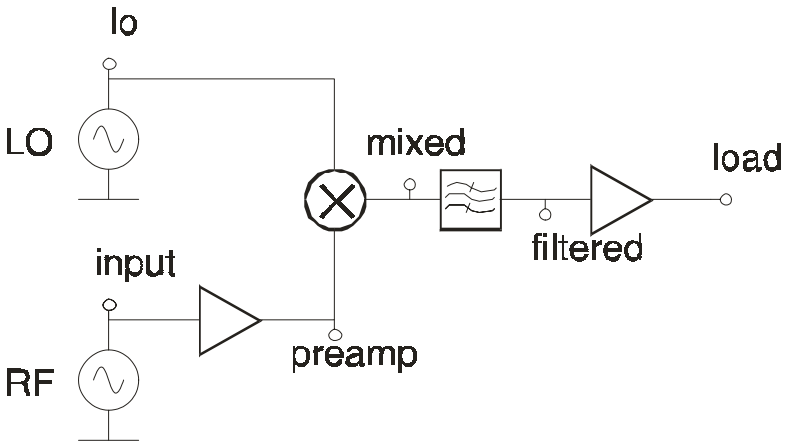
System Level Components:
These include complex system-level block description models for developing higher level system designs.

5 Simulation

Extensive modeling capabilities of VHDL-FD language are useless without effective simulation tools. Rincon offers not only a proven approach to model building, but also simulates circuit as well. While Rincon's initial focus provides HB simulation capabilities, other solvers will be available in the future that utilize this VHDL-FD based Model Library Approach.

5.1 System simulation

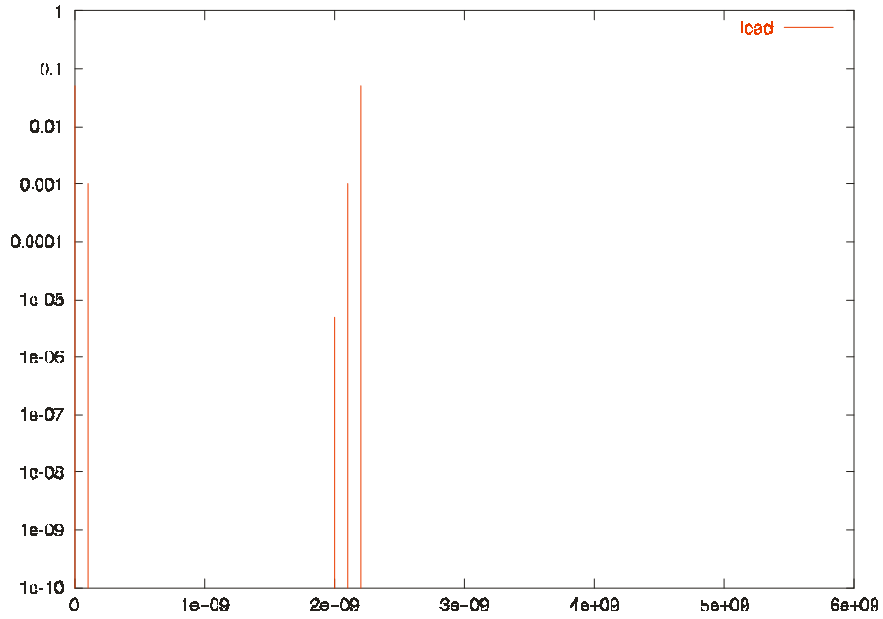
This example provides using the VHDL-AMS language and Rincon simulator to investigate system properties. Consider receiver system front-end with the following topology:



Code of VHDL-FD model is the following:

```
entity receiver is
end entity receiver;
-- model of simple receiver front-end with pre-amplefier,
-- mixer, low-pass filter and IF amplifier. All components
-- are represented in system level
architecture system of receiver is
  quantity input, preamp, lo, mixed, filtered, load: real;
  constant SourceFrequency1 : real := 1.e9;
  constant SourceOrder1 : real := 3;
  constant SourceFrequency2 : real := 1.1e9;
  constant SourceOrder2 : real := 5;
  constant SpectrumShape : real :=2; --
  constant lowfr : real := 2*SourceFrequency2;
begin
RF: entity source(eq)
  generic map(1.e-6, SourceFrequency1) port map (input);
PREAMP: entity amp (eq)
  generic map (1000) port map (input, preamp);
LO: entity source(eq)
  generic map(100.e-3, SourceFrequency2) port map (lo);
MIXER: entity multiplier(eq2)
  port map (preamp, lo, mixed);
FILTER: entity lpfilter(eq)
  generic map (lowfr) port map( mixed, filtered);
IFAMP: entity amp (eq)
  generic map (10) port map (filtered, load);
end;
```

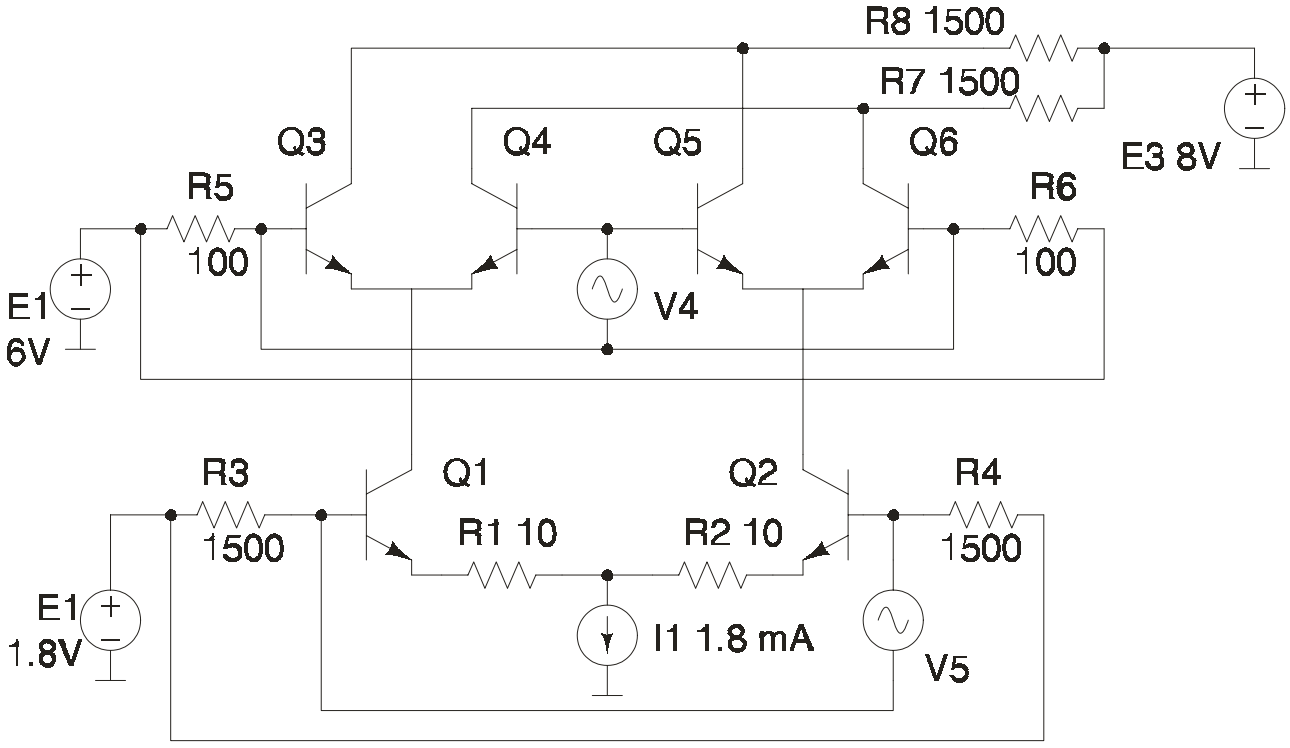
Simulating this system under 2-tone excitation gives us the following spectrum at the output:



Main components of spectrum are: DC, 2*LO, IF (LO-RF), LO+RF, 2*RF.

5.2 Circuit simulation

Consider one more VHDL-FD example in circuit simulation. Gilbert cell mixer schematics:

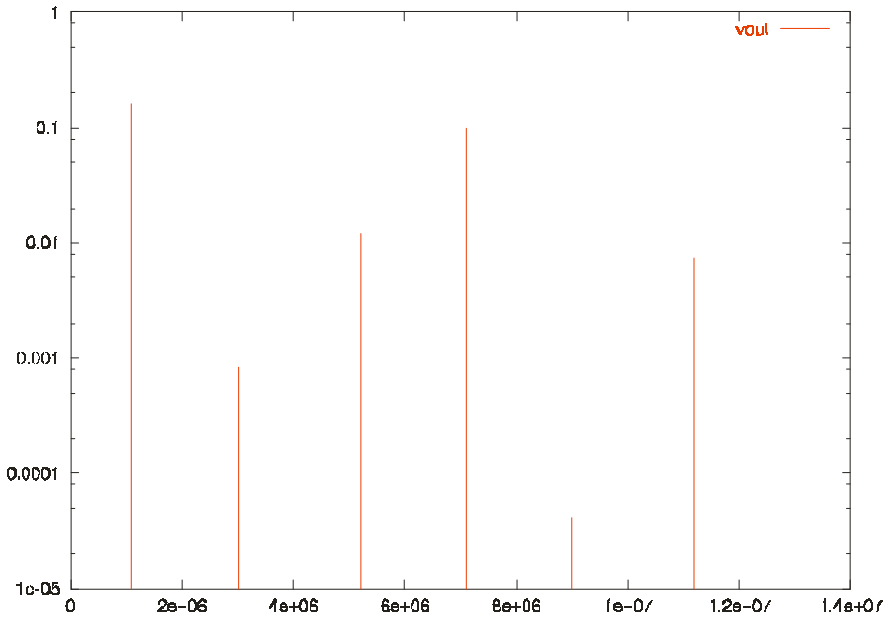


Circuit description will be the following:

```
entity gilbert is end entity gilbert;
architecture componental of gilbert is
-- setting predefined constants
```

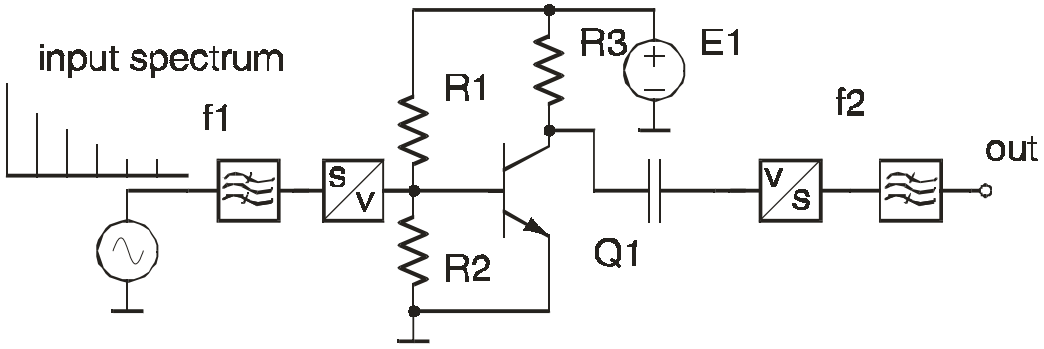
```
  constant SourceFrequency1 : real := 3.0e6;
  constant SourceOrder1 : real := 3;
  constant SourceFrequency2 : real := 4.1e6;
  constant SourceOrder2 : real := 3;
  constant SpectrumShape : real :=2;
-- define terminals
  terminal t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,
           t12,t13, t14 : electric;
  quantity vin across t9 to t10;
  quantity vout across t5 to t6;
  quantity k: real;
begin
  E1: entity DCG (eq) generic map (1.8) port map (t14);
  E2: entity DCG (eq) generic map (6.0) port map (t8);
  E3: entity DCG (eq) generic map (8.0) port map (t7);
  R5: entity res(eq)  generic map (100) port map (t8, t2);
  Q3: entity bjt(em1) port map (t3,t2,t5);
  Q4: entity bjt(em1) port map (t3,t1,t6);
  Q5: entity bjt(em1) port map (t4,t1,t5);
  Q6: entity bjt(em1) port map (t4,t2,t6);
  R6: entity res(eq) generic map (100) port map (t2, t8);
  LO: entity vsource(eq) generic map (1.0, SourceFrequency2)
      port map (t1, t2);
  R8: entity res(eq) generic map (1500) port map (t5, t7);
  R7: entity res(eq) generic map (1500) port map (t6, t7);
  R3: entity res(eq) generic map (1500) port map (t14, t9);
  Q3: entity bjt(em1) port map (t11,t9,t3);
  R1: entity res(eq) generic map (10) port map (t11, t13);
  R4: entity res(eq)
      generic map (1500) port map (t14, t10);
  Q3: entity bjt(em1) port map (t12,t10,t4);
  R2: entity res(eq) generic map (10) port map (t12, t13);
  Ra: entity res(eq) generic map (10000) port map (t13, ground);
  I1: entity DCI(eq) generic map (1.e-3) port map (t13, ground);
  RF: entity vsource(eq) generic map (10.e-3, SourceFrequency1) port
map (t9, t10);
-- single equation to "detect" output voltage
  k == vout;
end architecture;
```

This circuit has been simulated under different conditions (different biases and V4 and V5 voltages). Some results are presented in [1]. Here is presented output spectrum - voltage between collectors of Q5 and Q6 BJTs.



5.3 Mixed mode examples

One more example is intended to show the ability of VHDL-FD to simulate mixed system/circuit level projects. Consider 1-stage transistor amplifier surrounding filters. From the front-end it is driven with some kind of periodic excitation, then it is filtered, then amplified and filtered once more. Only amplifier will be represented with circuit, other parts will be represented with system-level components.



Here s/v and v/s are "signal/voltage" and "voltage/signal" converters with internal output and input impedances respectively. Their VHDL-FD code may look like:

```
entity sv_converter is
  generic (k, Ri: real);
  port (quantity inp: real;
        port outp: electrical);
end entity;
-- this entity converts abstract "inp" into voltage at port outp
architecture eq of sv_converter is
  quantity v across i through outp;
begin
  (v-k*inp)==i*Ri; -- e.m.f = k*inp; internal resisteance = Ri;
end architecture;
```

And voltage-signal converter:

```
entity vs_converter is
  generic (k, Ri: real);
  port (port inp: electrical;
        quantity outp: real);
end entity;
-- this entity converts input volatge into signal
architecture eq of vs_converter is
  quantity v across i through inp;
begin
  v==i*Ri;   -- input resistance equation
  outp==k*v; -- define output value
end architecture;
```

Source with spectrum may be interesting too:

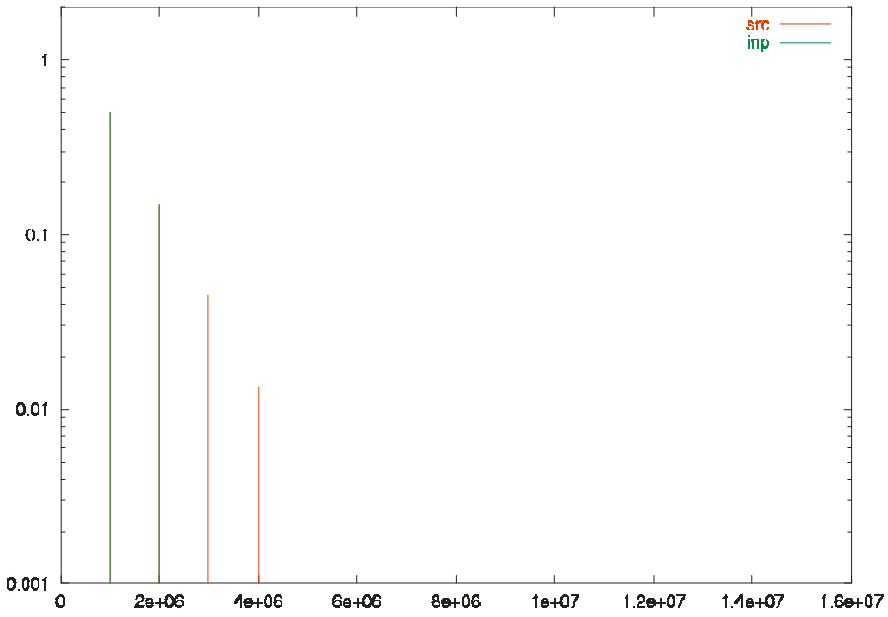
```
entity sp_source is
  generic (fr, val, decr: real);
  port (quantity out: real);
end entity sp_source;
architecture eq of sp_source is
  if(FREQUENCY() == fr) use
    out'FD == val;
  else if (FREQUENCY() == 2*fr) use
    out'FD == COMPLEX(val*decr,0); -- decr is less than 1
  else if (FREQUENCY() == 3*fr) use
    out'FD == COMPLEX(val*decr**2,0); -- third harmonic
```

```
  else if (FREQUENCY() == 4*fr) use
    out'FD == COMPLEX(val*dect**3,0); -- fourth harmonic
  else
    out'FD == ZERO; -- ZERO is predefined
  end use;
end architecture;
```

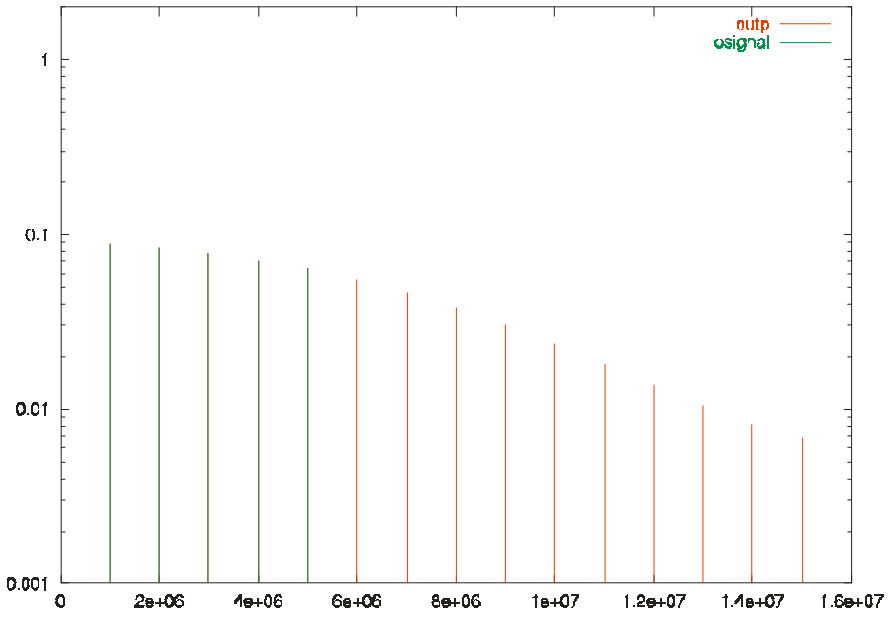
As all components are defined already, the code will be:

```
entity mixed is end entity;
--
architecture comp of mixed is
  quantity src, f1, f2, osignal: real;
  terminal inp, dc, collector, outp: electrical;
  constant SourceFrequency1 : real := 1.0e6;
  constant SourceOrder1 : real := 10;
  constant SpectrumShape : real :=1;
  constant fr2: real :=2*SourceFrequency1;
  constant fr5: real :=5*SourceFrequency1;
begin
 SRC: entity sp_source(eq) generic map(SourceFrequency1, 1, 0.5) port
map(src);
  F1: entity lpfilter(eq) generic map(fr2) port map (src, f1);
  SV: entity sv_converter(eq) generic map(1,0.1) port map(f1, inp);
  R1: entity res(eq) generic map (6000) port map (dc, inp);
  R2: entity res(eq) generic map (420) port map (inp, ground);
  R3: entity res(eq) generic map (1000) port map (dc, collector);
  Q1: entity bjt(em1) port map (ground,inp,collector);
  C1: entity cap(eq) generic map(1.e-6) port map (collector,outp);
  DC: entity DCG (eq) generic map (10) port map (dc);
  VS: entity vs_converter(eq) generic map(1,100) port map(outp, f2);
  F2: entity lpfilter(eq) generic map(fr5) port map (f2, osignal);
end architecture;
```

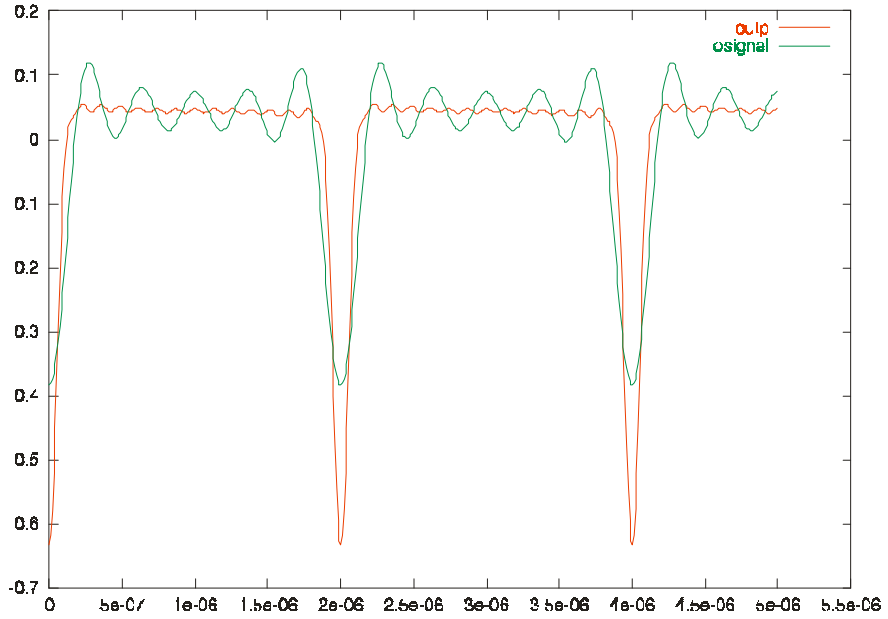
Simulation gives the following results. Spectra at input and after filter1:



Spectra after capacitor and at the output:



Waveform at the output:



This shows usefulness VHDL-FD modeling language as mixed system/circuit simulation tool.

6 Conclusion

Rincon represents a rigorous tool for frequency domain modeling and simulation. Together with the VHDL-FD language, it allows modeling of both lumped nonlinear devices and distributed linear with complex behavior w.r.t. frequency. Flexible approach does not limit designers and makes possible to simulate complex circuits and/or systems with different degree of detail.

7 References

[1] G. Serdyuk, D. Goodman "VHDL...." CriMiCo'2001
[2] VHDL-AMS Language Reference Manual, IEEE.