Analogue and mixed-signal behavioural synthesis from VHDL-AMS

Tom J Kazmierski
Department of Electronic and Computer Science
University of Southampton, United Kingdom
tjk@ecs.soton.ac.uk, http://www.syssim.ecs.soton.ac.uk

Outline

- Introduction
  - modern design challenges
  - AMS design process
- Basics of AMS synthesis
- Analogue/AMS circuit synthesis - state of the art
- AMS behavioural synthesis from VHDL-AMS
- Introduction to VHDL-AMS (with digressions relevant to synthesis)
- VHDL-AMS synthesis methodologies and tools
  - NEUSYS
  - FIST
  - VASE
- High level synthesis – a modern approach
- Conclusion, looking into the future

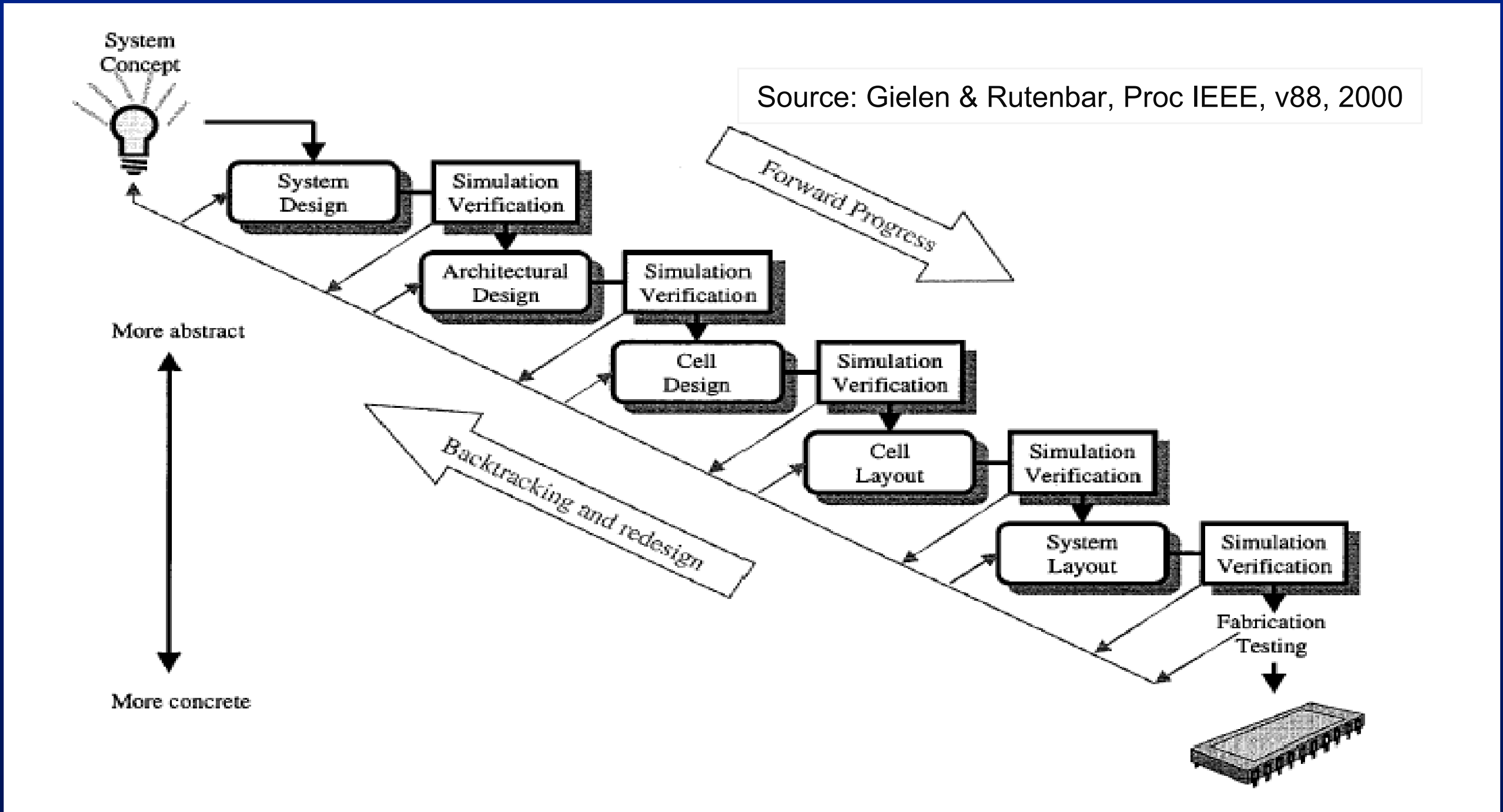
Design challenges today

- Ever increasing design complexity
  - Highly integrated SoC include analogue, digital, RF sections on one chip
  - Typical mixed SoC: MPEG-4 encoders, network processors, software radio.
  - Every year, the number of transistors per chip rises by 58%, whereas the design productivity increases by only 20%.

- There are functions that cannot be performed by digital systems
  - Processing of input signals from a sensor, microphone, antenna
  - Processing of output signals: actuators, transmission lines
  - Processing of very high frequency (RF) signals

Department of Electronics and Computer Science
Southampton University

AMS design process



IC synthesis

- Integrated circuit synthesis - process of automatic generation of integrated circuit layout masks from a high-level description.
- Behavioural specifications are usually written in a high-level hardware description language
- Classical two-stage synthesis process:
  - automatic translation of the behavioural description to a circuit structure, followed by:
  - silicon compilation to produce fabrication masks or FPGAs.
- Advances in IC technology have led to the growing popularity of mixed-signal ASICs, which comprise both analogue and digital circuit blocks.
- Nowadays digital designs are fully automated while the analogue part of a typical ASIC still needs to be designed manually

Typical AMS synthesis tasks

(1) Specification

(2) Architecture generation,

(3) Performance model generation

(4) Parameter exploration or constraint transformation.

- Most approaches particularize the general synthesis flow for specific types of analogue systems, such as filters, neural networks, DSP systems etc.

AMS synthesis lags behind digital synthesis

- With the advent of digital synthesis tools and semi-custom layout design techniques, analogue ASIC blocks may now consume 90% of the overall design time, while using only 10% of the silicon area.

- One of the obvious difficulties in analogue synthesis: the absence of a mixed-signal high-level hardware description language commonly accepted as a standard throughout the CAD industry.

AMS synthesis vs. digital synthesis

- ***Hierarchy.***
  - In digital systems - a widely accepted distinction of hierarchy levels.
  - No formal and well-defined hierarchical treatment for analogue system descriptions seems to exist.
- ***Performance Specification.***
  - Digital hardware and software are optimized for cost, speed, and power consumption.
  - In contrast, analogue and RF circuits must meet specific constraints, e.g. linearity, bandwidth, dynamic range, and image rejection - wide spectrum of performance characteristics often unique to an application.
    - E.g. application-related requirements for area, power, gain-bandwidth product, DC gain, phase and gain margin, slew rate, etc.
    - Specifications may lead to conflicting synthesis constraints
    - Some constraints, such as the gain-bandwidth product, might need to be specified in terms of design trade-offs.

Department of Electronics and Computer Science
Southampton University

...AMS synthesis vs. digital synthesis

- ***Device Sizing.***
  - Device sizing of analogue circuits - a significant effect on performance. E.g. changes in transistor sizes will usually have a major impact on the dynamic behaviour.
- ***Standard Cells.***
  - Analogue designs - wide range of primitive circuit blocks, many of which perform limited functions appropriate only for certain applications.
  - Digital circuits are constructed with a smaller number of cells, which are easy to characterise and standardise.

...AMS synthesis vs. digital synthesis

- ***Technology Influence.***
  - Performance of analogue circuits - very sensitive to technology and environmental parameters.
  - Crucial role of precision device matching and modelling in analogue designs.
  - Importance of layout - minimum area is not the sole concern as it is often the case in digital circuit synthesis.
- ***System-level Interactions.***
  - In analogue circuits - perturbations at system level can be as destructive as they are on the microscopic level.
  - Integration with digital circuitry for mixed-signal application would need careful consideration.

...AMS synthesis vs. digital synthesis

- ***High Performance Applications.***
  - Nowadays, designers resort to analogue solutions largely in high-performance designs,
  - In contrast, most digital designs are used in applications requiring moderate performance specifications.

AMS synthesis - state of the art

	Name	Type	Origin and description	Year
1	IDAC	KB	Swiss Center for Electronics and Microtechniques, Switzerland; Users select a topology from a library. 1991: open tool for design reuse.	1987, 1991
2	AN-COM	KB	General Electric Company, New York, USA Domain knowledge is used for successive decomposition of circuit specification.	1988
3	CAMP	KB	Univ. of Southern California, USA; Uses iterative self-reconstructing technique and circuit simulation for a flexible architecture.	1988
4	DELIGHT SPICE	OB	Harris Corporation, USA; Utilises a SPICE simulator as the optimisation core .	1988
5	OASYS	KB	Carnegie Mellon Univ., USA Top-down hierarchical structure in knowledge application	1989
6	BLADES	KB	AT&T Bell Labs., USA Uses artificial intelligence to combine formal and intuitive knowledge.	1989

Type: KB – knowledge based, OB – optimisation based, MS – mixed-signal.

...AMS synthesis - state of the art

	Name	Type	Origin and description	Year
7	OPASYN	OB	Univ. of California, Berkeley, USA Silicon compilation of op amps .	1990
8	ASAIC	OB	Katholieke Universiteit Leuven, Belgium; Features a symbolic analysis programme, ISAAC and an optimiser, OPTIMAN	1990
9	CHIPAIDE	KB	Imperial College, UK Uses a hierarchical approach to produce first-cut circuit topology	1990
10	OAC	OB	Kyoto Univ., Japan CMOS op amp compiler which runs a simulation-based optimiser as a post-processor.	1990
11	STAIC	OB	Univ. of Waterloo, Ont., Canada Uses a description language in its multilevel modelling scheme. Synthesis uses successive solution refinement technique	1992
12	MINLP-Maulik	OB	Carnegie Mellon Univ., USA Allows simultaneous circuit topology and parameter selection	1992

Type: KB – knowledge based, OB – optimisation based, MS – mixed-signal.

...AMS synthesis - state of the art

	Name	Type	Origin and description	Year
13	ARCH-GEN	MS	Vanderbilt Univ., USA Synthesis of filter systems from behavioural specifications	1995
14	KANDIS	MS	Johann Wolfgang Goethe Univ., Frankfurt Translates hybrid-VHDL into an intermediate representation (KIR graph), which is then used by a high-level synthesis tool and an estimator.	1995
15	ASTRX/ OBLX	OB	Carnegie Mellon Univ., USA Uses asymptotic waveform evaluation (AWE) to evaluate circuit performance and simulated annealing for optimisation	1996
16	VASE	MS	Univ. of Cincinnati, USA Behavioural VHDL-AMS specifications are compiled to obtain a hierarchical intermediate representation. An architecture generator and performance estimator is part of the synthesis environment.	1997
17	NEUSYS and FIST	MS	Univ. of Southampton, UK The architecture generator uses parse trees obtained from behavioural VHDL-AMS specifications and a primitive block library.	2000 2002

Type: KB – knowledge based, OB – optimisation based, MS – mixed-signal.

VHDL-AMS and synthesis

- In 1999 IEEE DASC adopted the 1076.1 standard for VHDL with analogue extensions informally called VHDL-AMS
- VHDL-AMS extends the modelling power of VHDL to the domain of continuous and discrete-continuous systems.
- VHDL-AMS is expected to advance behavioural modelling, and consequently, automated synthesis of analogue and mixed-signal systems.

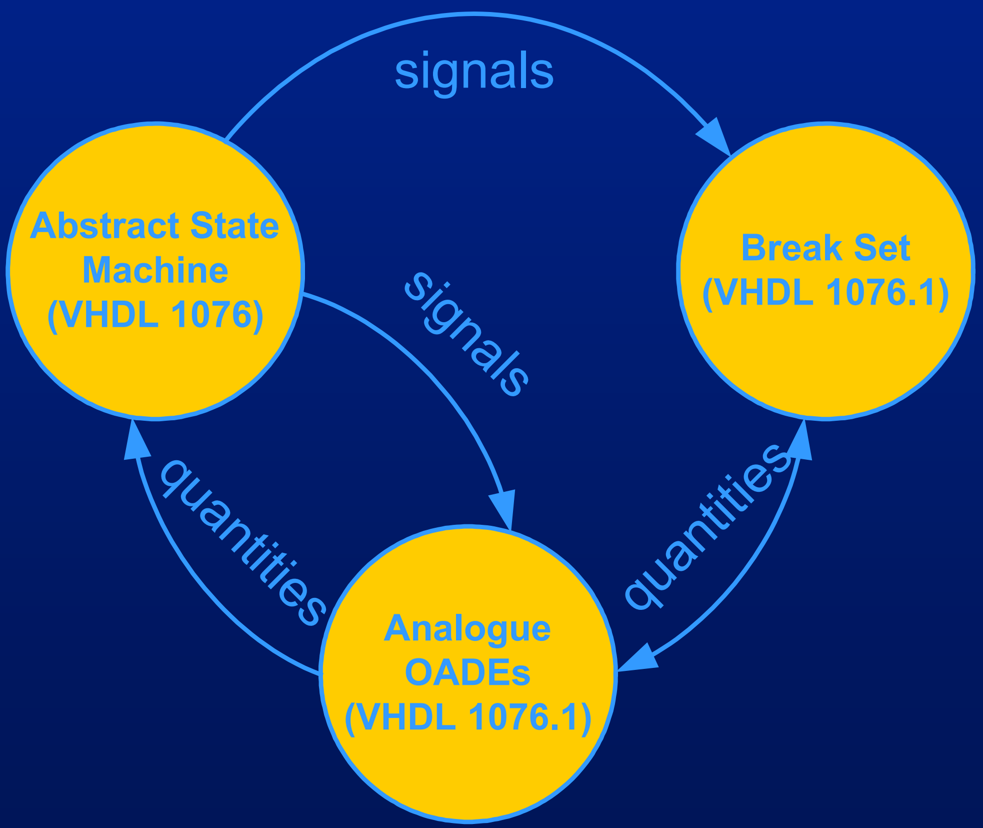
Department of Electronics and Computer Science
Southampton University

Basic AMS concepts in VHDL-AMS

- Topology models of networked physical systems, such as Kirchhoff's laws for electrical circuits
- Abstract models of physical components, such as non-linear differential and algebraic equations
- Abstract models of interfaces between the discrete (digital) and continuous (analogue) domains
- Models for frequency-domain and noise simulations
- A VHDL-AMS simulator supports mixed, event-driven and continuous behavior

Department of Electronics and Computer Science
Southampton University

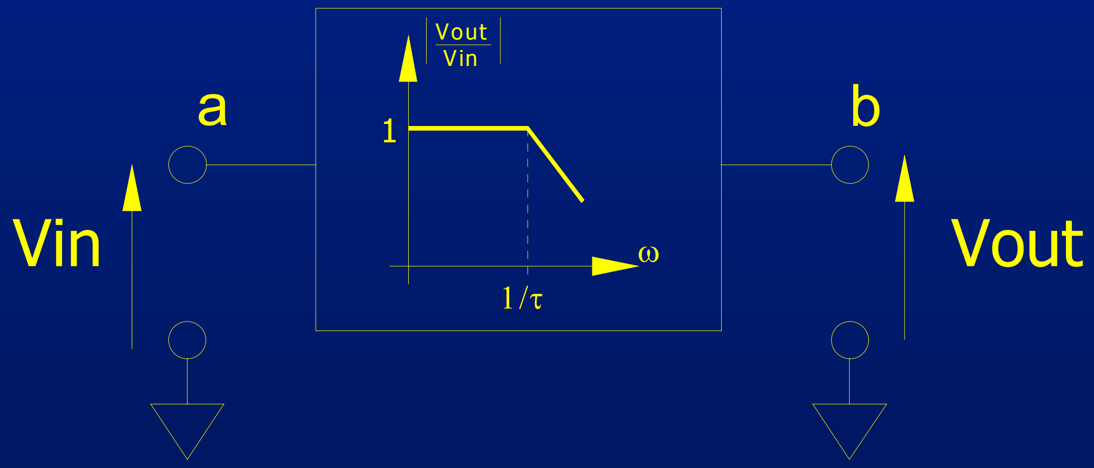
VHDL-AMS concepts



Department of Electronics and Computer Science
Southampton University

Simple analogue behaviour

Low-pass filter



$$\tau \frac{dVout}{dt} + Vout = Vin$$

Low-pass filter - time-domain model

```
entity LowPass is
  generic ( tau: time := 1.0E-6);
  port ( quantity Vin: voltage;
         quantity Vout: out voltage);
end entity LowPass;

architecture DifferentialEqn of LowPass is
begin
  Vin == tau*Vout'DOT + Vout;
end architecture;
```

Simultaneous statement

Department of Electronics and Computer Science
Southampton University

Low-pass filter - s-domain model

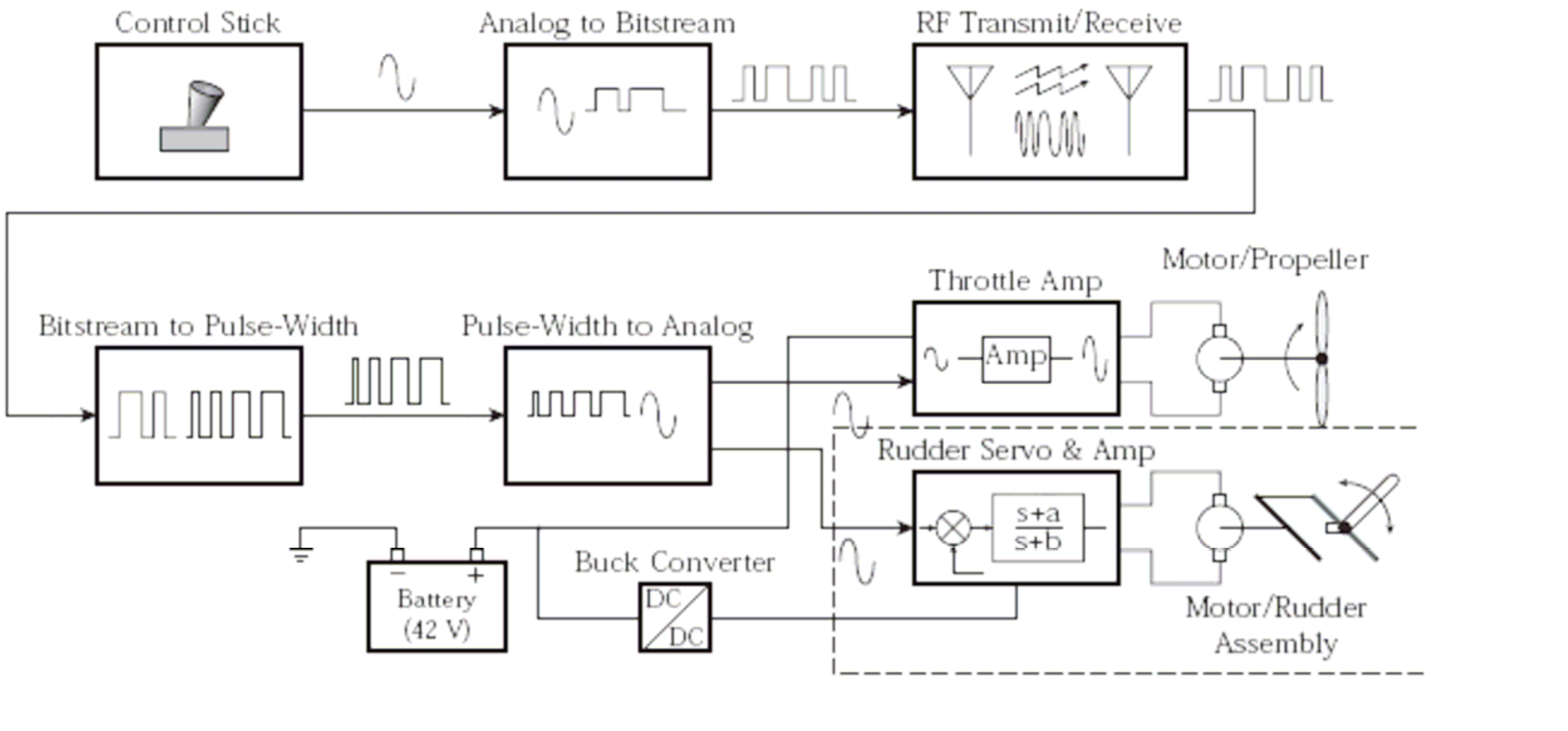
```
entity LowPass is
   generic ( tau: real := 1.0E-6);
   port ( quantity Vin: voltage;
             quantity Vout: out voltage);
end entity LowPass;

architecture Transfer of LowPass is
 constant num: real_vector :=(0=>1.0);
 constant den: real_vector := (tau,1.0);
begin

   Vout == Vin'LTF(num,den);

end architecture;
```

transfer function:

$$\frac{1}{\tau s + 1}$$

Department of Electronics and Computer Science
Southampton University

Complex mixed-domain system – airplane control (source: Mentor Graphics)



RC Airplane System with Rudder System highlighted.

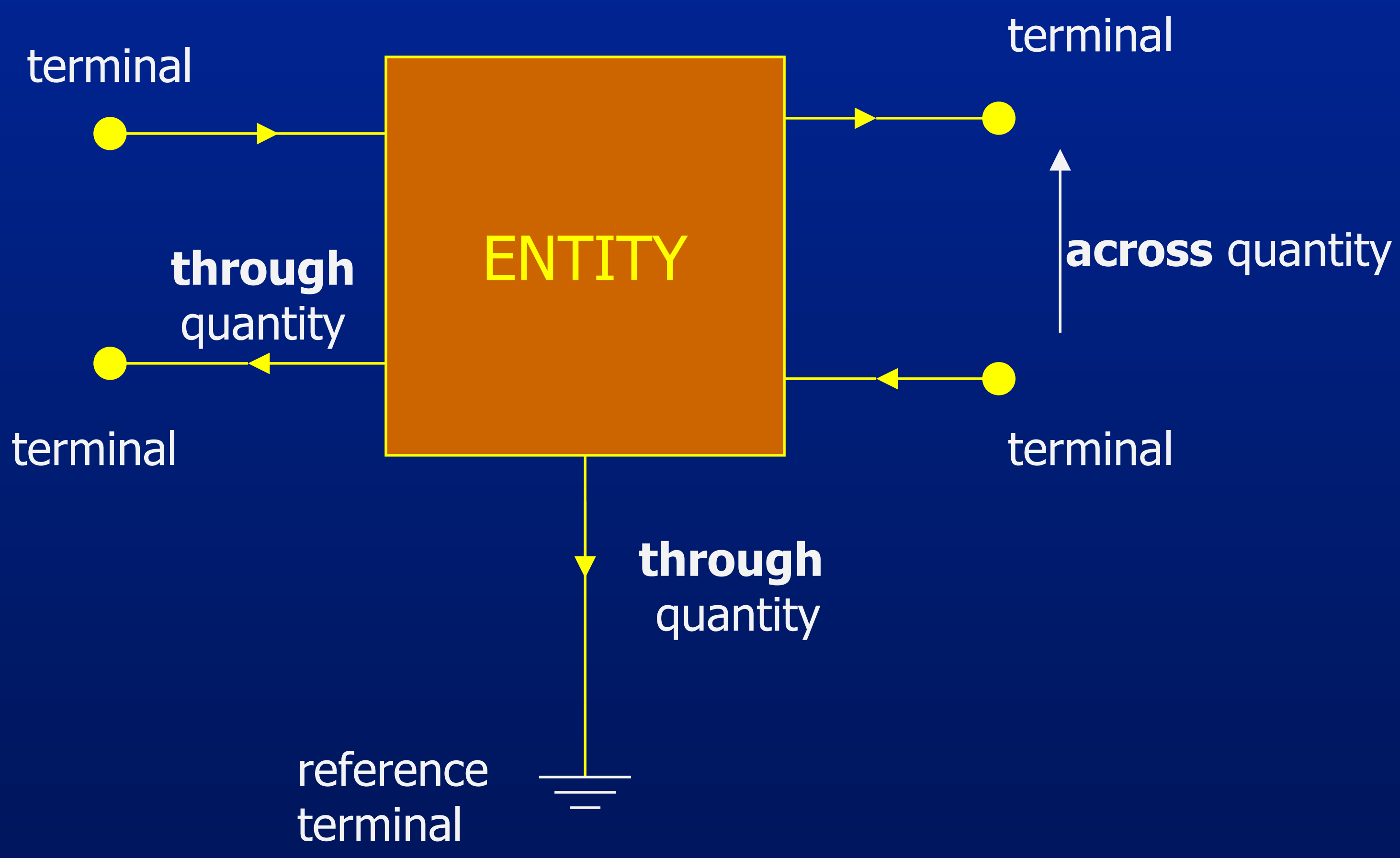
Quantities and simultaneous statements

- **quantity**
  - unknown in the analogue equation set
  - continuous-time waveform
- **simultaneous statement**
  - analogue constraint (equation)

- The VHDL-AMS simulator evaluates quantities such that the constraints specified by the simultaneous statements are satisfied with certain accuracy
- Analogue accuracy is controlled by user-specified or the default tolerances

Modelling of networked physical systems, i.e. systems with topology

- **nature**
  - supports descriptions of structured (networked) physical systems
- **terminal**
  - connectivity point in a structured system
- **across quantity, through quantity**
  - support flow- and effort-like quantities in structured systems
  - provide system topology information and enable the underlying simulator to form topology (Kirchoff's) constraints

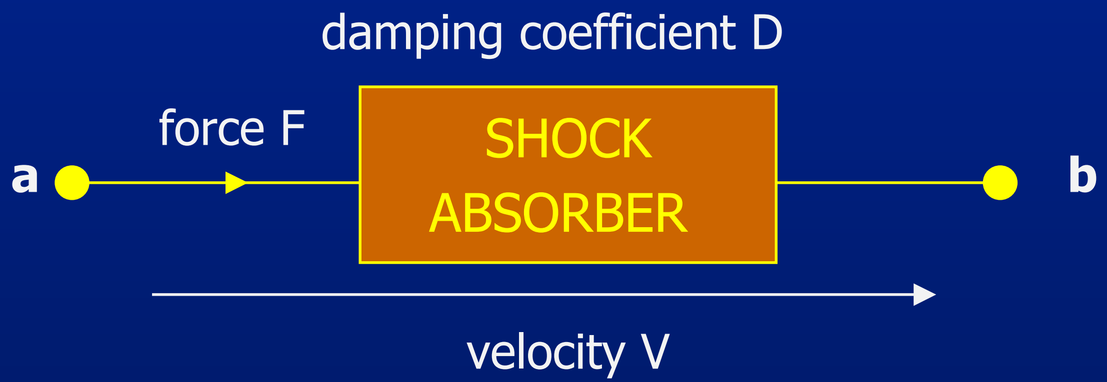

terminal
terminal
ENTITY
through
quantity
across quantity
terminal
terminal
through
quantity
reference
terminal

```
package ElectricalDomain is
   subtype voltage is real range –1.0E4 to 1.0E4 tolerance "voltage";
   subtype current is real range –1.0E2 to 1.0E2 tolerance "current";

   nature electrical is voltage across
                        current through
                        ground reference;

   nature electrical_vector is
          array (natural range <>) of electrical;

   alias undefined is real'LOW; -- undefined value used in
                                -- some SPICE-like models
   -- some physical constants
  constant Boltzmann : real :=1.380662e-23; -- Boltzmann constant
  constant ElectronCharge : real :=1.6021892e-19; -- electronic charge
  shared variable TEMP0: real := 300.0; -- ambient temperature

end package ElectricalDomain;
```

Department of Electronics and Computer Science
Southampton University

```
package MechanicalDomain is

  -- model of transitional mechanics
  subtype velocity is real tolerance "velocity";
  subtype force is real tolerance ″force";
  nature mechanical is velocity across force through chassis reference;

  -- model of rotational mechanics
  subtype angular_velocity is real tolerance ″angular_velocity";
  subtype torque is real tolerance ″torque";
  nature rotational is angular_velocity across torque through
                              rotational_ref reference;

  -- model of liquid mechanics
  subtype pressure_head is real tolerance ″preassure_head";
  subtype volume_flow is real tolerance ″volume_flow";
  nature liquid is pressure_head across volume_flow through tank reference;

end package MechanicalDomain;
```

Mechanical entity example

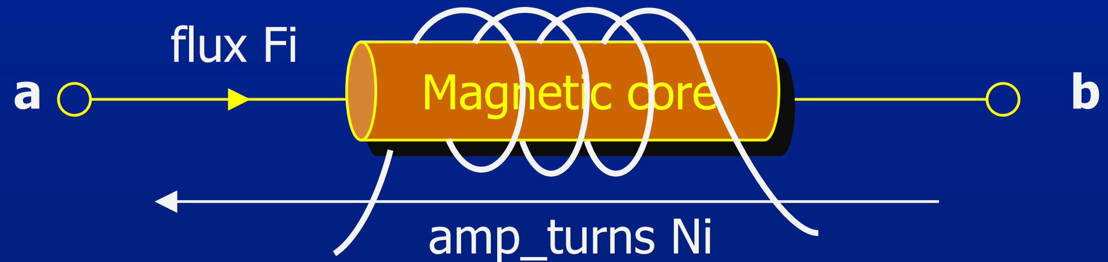



```
entity shock is
   generic (D: real := 0.5); -- Ns/m
   port (terminal a,b: mechanical);
end entity shock;
```

```
architecture damper of shock is
  quantity V across F through a to b;
begin
    V == F*D;
end architecture damper; -- of shock
```

Department of Electronics and Computer Science
Southampton University

```
package MagneticDomain is

   subtype amp_turns is real tolerance "current";
   subtype flux is real tolerance "flux";

   nature magnetic is amp_turns across flux through
                            mag_ref reference;

end package MagneticDomain;
```

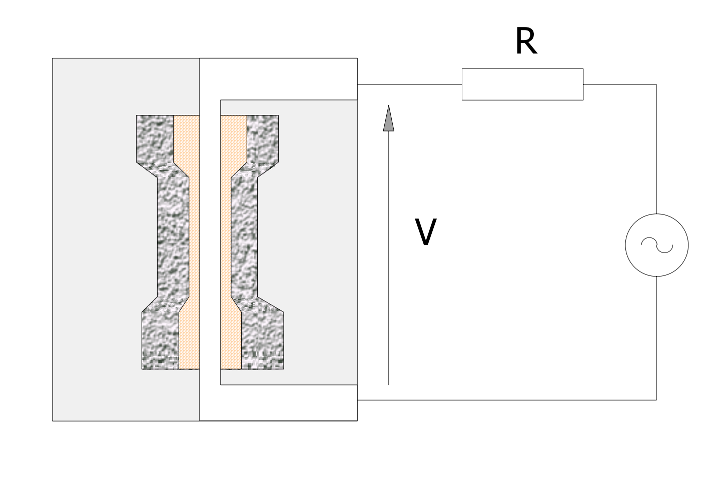


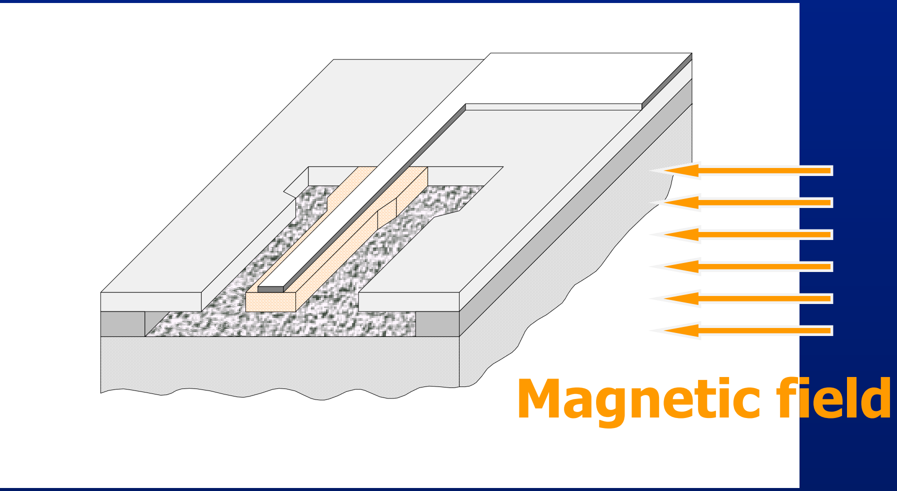

```
entity core is
   generic (reluctance: real := 1.0E-3); -- Wb/A
   port (terminal a,b: magnetic);
end entity core;
```

```
architecture MagneticOhmsLaw of core is
  quantity Ni across Fi through a to b;
begin
    Fi == reluctance*Ni;
end architecture MagneticOhmsLaw; -- of core
```

Micro-electromechanical system

Vibrating silicon beam

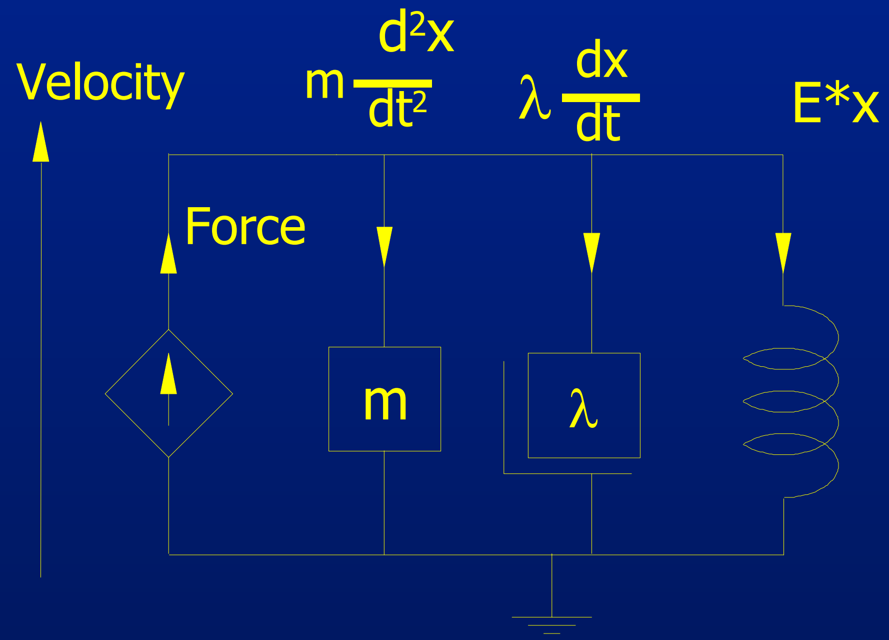




Department of Electronics and Computer Science
Southampton University

Vibrating micro-beam

Mechanical model



m – mass
x – displacement
λ- damping factor
E – Young's modulus

```
-- excitation force on beam from magnetic field:
    Force := Iab*length*B;
```

Department of Electronics and Computer Science
Southampton University

Vibrating micro-beam
Entity declaration

```
entity SiliconMicroBeam is
  generic ( length: real := 2.0E-3;
            height: real := 6.0E-6;
            breadth: real := 100.0E-6;
            lambda:real:= 0.05; -- damping factor
            B: real := 0.145; -- magnetic induction
            Rs: real := 0.1); -- electrical resistance
  port (terminal a,b :electrical);
end entity SiliconMicroBeam;
```

Vibrating micro-beam

Architecture declaration (I)

```
architecture Electromechanical of SiliconMicroBeam is
    constant roSi: real := 2.3E3; -- Silicon density
    constant E:real := 130.0E9; -- effective Young's modulus
    constant betaL:real := 4.73; -- vibration coeff for double-ended beam
   -- mechanical circuit
    terminal beam: mechanical;
    quantity Velocity across
                 Force through beam to chassis;
    quantity x: real; -- mechanical displacement
  -- electrical circuit
    quantity Vab across Iab through a to b;
begin


   ...
```

Department of Electronics and Computer Science
Southampton University

Vibrating micro-beam

Architecture declaration (II)

```
-- (cont)
  procedural is
     variable In,omega : real; -- momentum factor and res. frequency
   begin
     In := breadth*height**3/12.0; -- momentum factor
    -- resonant frequency
     omega := betaL**2*sqrt(E*In/(roSi*breadth*height*length**4));
    -- excitation force on beam from magnetic field
     Force := Iab*length*B;
   -- mechanical vibration equations
     x := Force/E - Velocity'DOT/omega**2 - lambda/E*Velocity;
     Velocity := x'DOT;
  end procedural;
  -- electrical circuit equation
   Vab == Iab*Rs;
end architecture  Electromechanical;
```

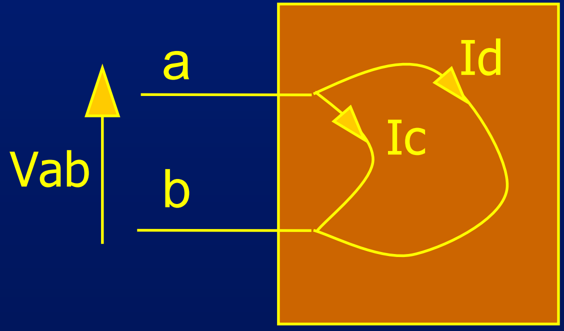
Quantities vs. signals

- A quantity represents a continuous-time waveform, a signal is a discrete-time (event list) waveform
- A quantity is an unknown in the set of simultaneous differential-algebraic equations; a signal is driven by VHDL processes
- A scalar quantity must be of a floating-point type, signals can have bit, enumerated, integer or floating-point values

Department of Electronics and Computer Science
Southampton University

Implicit network equations

quantity Vab **across** Id,Ic **through** a **to** b;



Implicit KVL equation:

Vab == a - b

Implicit KCL equations:

a'contribution == Id+Ic

b'contribution == -(Id+Ic)

Implicit quantities (examples)

Q'DOT	time derivative of Q
Q'INTEG	time integral of Q from time=**0** to **now**
Q'DELAYED(T)	value of Q at time=**now**-T

- the use of Q'DOT and Q'INTEG gives rise to differential and integral equations

Ordinary differential equations (ODEs) in VHDL-AMS

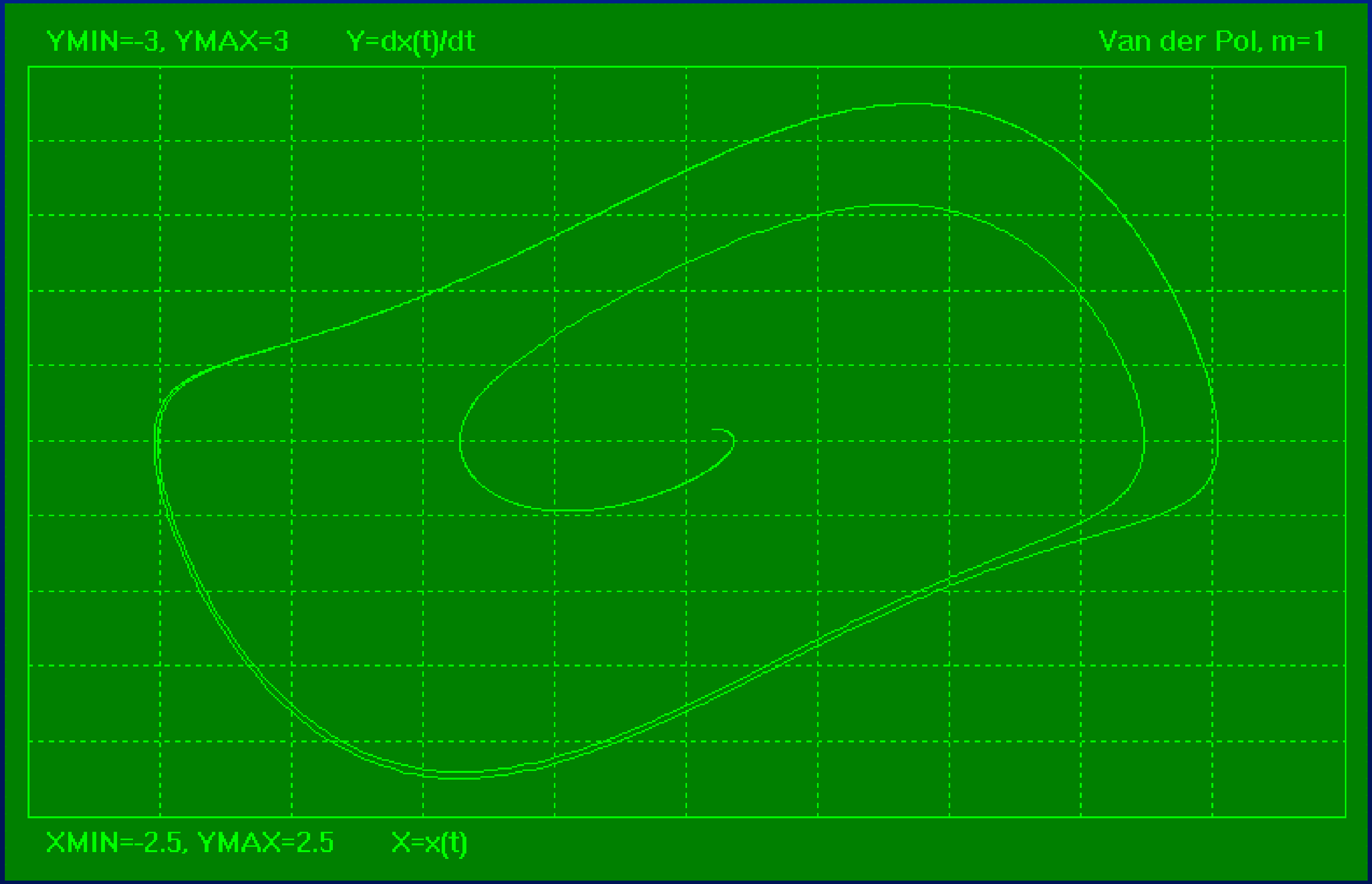
Van der Pol oscillator:

$$\ddot{x} - m(x^2 - 1)\dot{x} + x = 0$$

```
entity VanDerPol is
  generic (m: real := 1.0);
  port ( quantity x: out real := 0.1);
end entity;
```

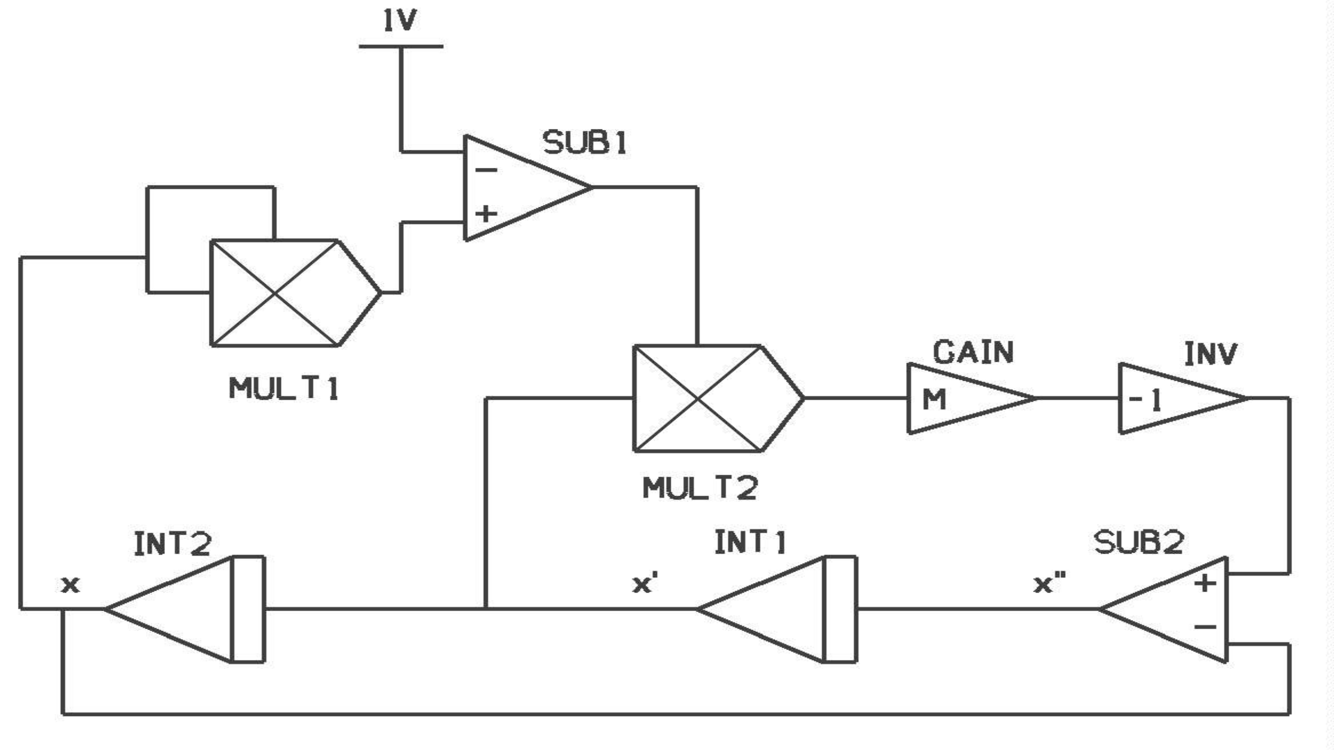
```
architecture behaviour of VanDerPol is
begin
      x == -x'DOT'DOT+m*(1.0-x*x)*x'DOT;
end architecture;
```

Van der Pol oscillator



Department of Electronics and Computer Science
Southampton University

Synthesis of Van der Pol oscillator
opamp-level structure generated by NEUSYS



Department of Electronics and Computer Science
Southampton University

A system of exotic ODEs

Lorenz Chaos:

$$\dot{x} = s(x-y)$$

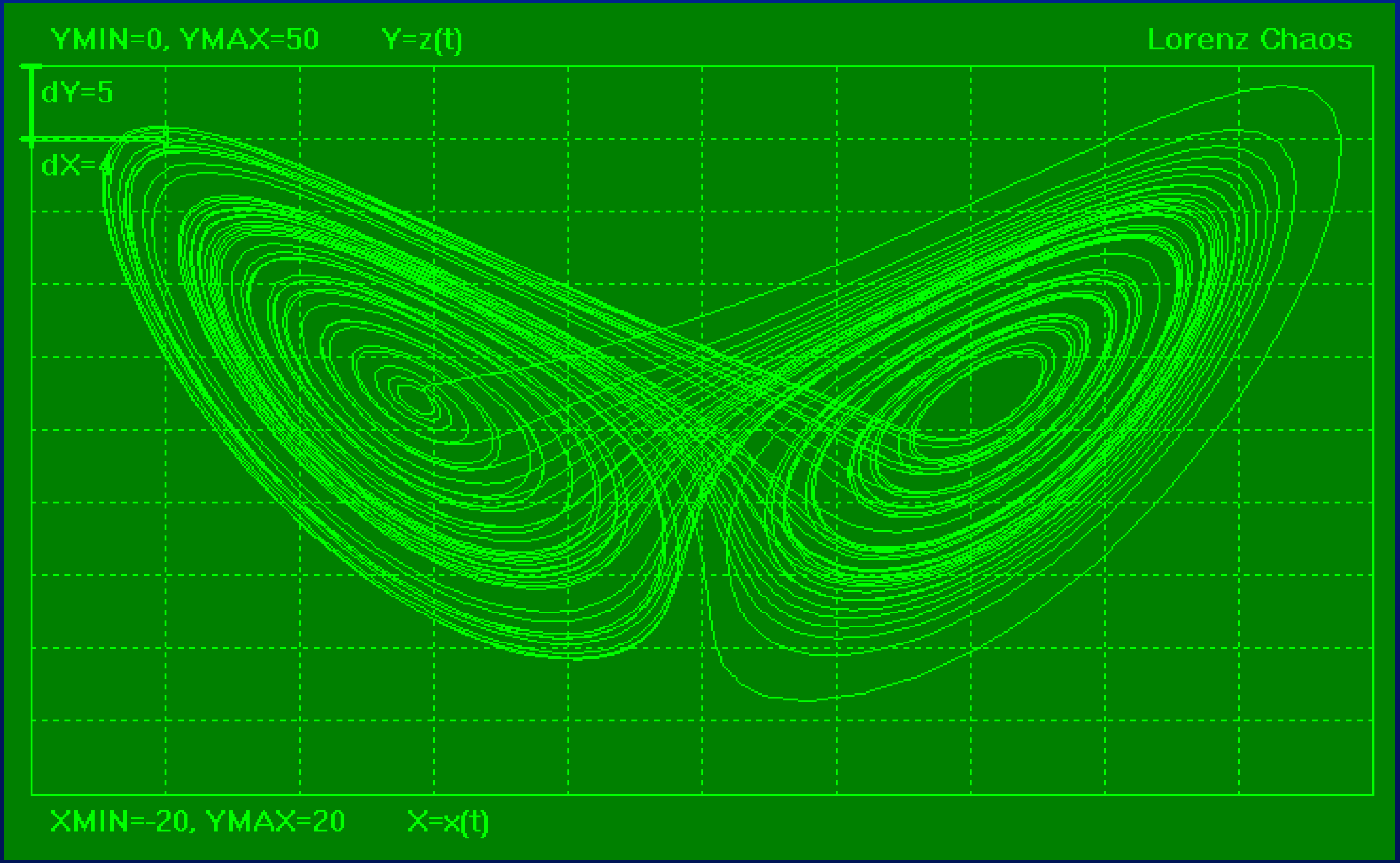
$$\dot{y} = rx - y - xz$$

$$\dot{z} = xy - bz$$

```
entity LorenzChaos is
  generic (s: real := 10.0,
           b: real := 2.667,
           r: real := 28.0);
  port ( quantity x: out real := 0.0,
                  y: out real := 5.0,
                  z: out real := 25.0);
end entity;
```

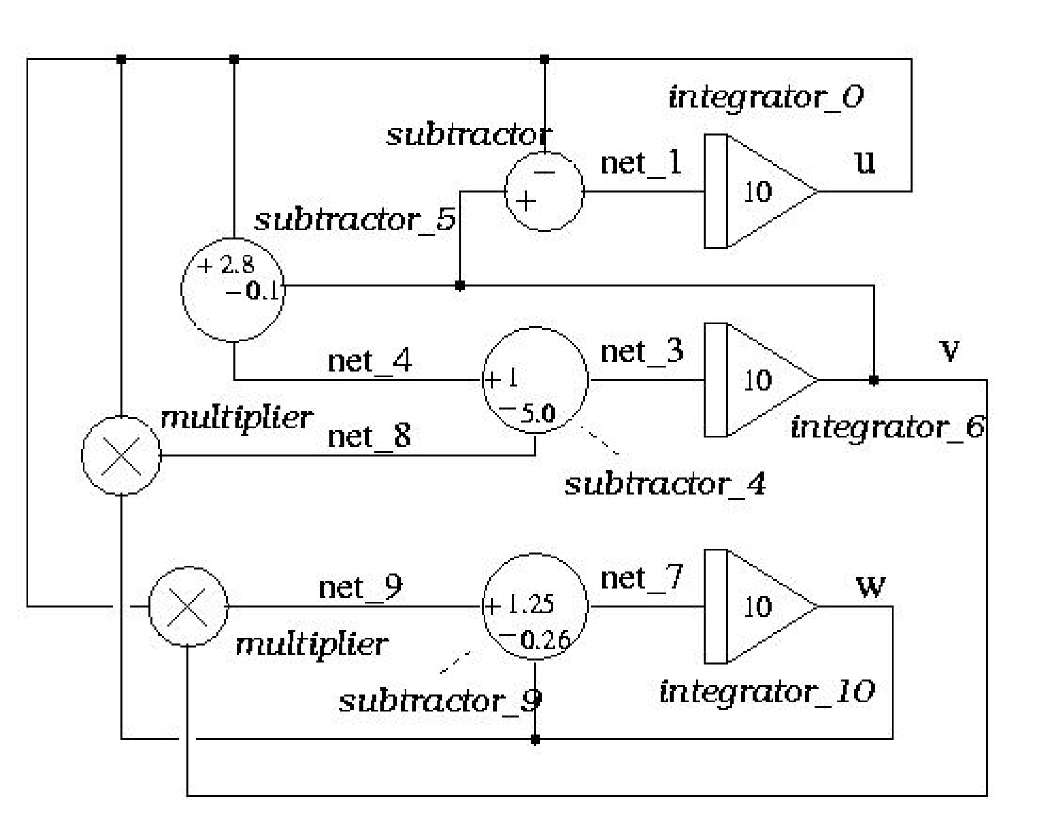
```
architecture behaviour of LorenzChaos is
begin
        x'DOT == s*(x-y);
        y'DOT == r*x-y-x*z;
        z'DOT == x*y - b*z;
 end architecture;
```

Lorenz Chaos – y(t) vs. x(t)



Department of Electronics and Computer Science
Southampton University

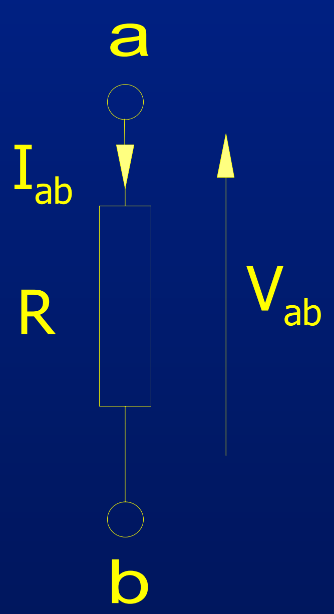
NEUSYS synthesis of Lorenz Chaos,
Opamp level diagram of synthesised circuit



Simultaneous statements

- Simultaneous statements represent the analog part of the VHDL-AMS model
- They express explicit algebraic and differential equations
- The analogue kernel calculates the values of quantities from the constraints specified by the simultaneous statements and implicit (topology) equations

Simple simultaneous statement

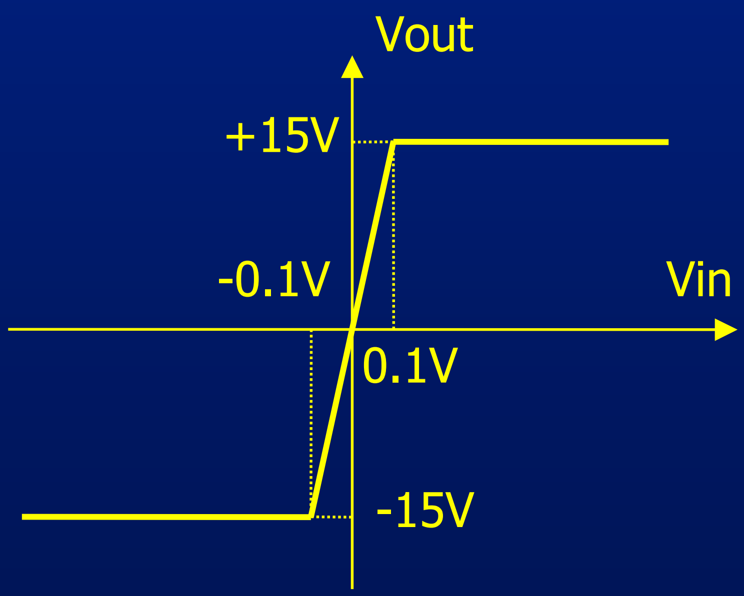



```
entity resistor is
   generic (R:real := 0.0);
   port (terminal a,b: electrical);
end entity resistor;
--
architecture OhmsLaw of resistor is
   quantity Vab across Iab through a to b;
begin
   -- Ohm's Law
   Vab == Iab*R;
end architecture OhmsLaw;
```

Composite simultaneous statements

- Simultaneous **if** statement
- Simultaneous **case** statement
- Simultaneous **procedural** statement

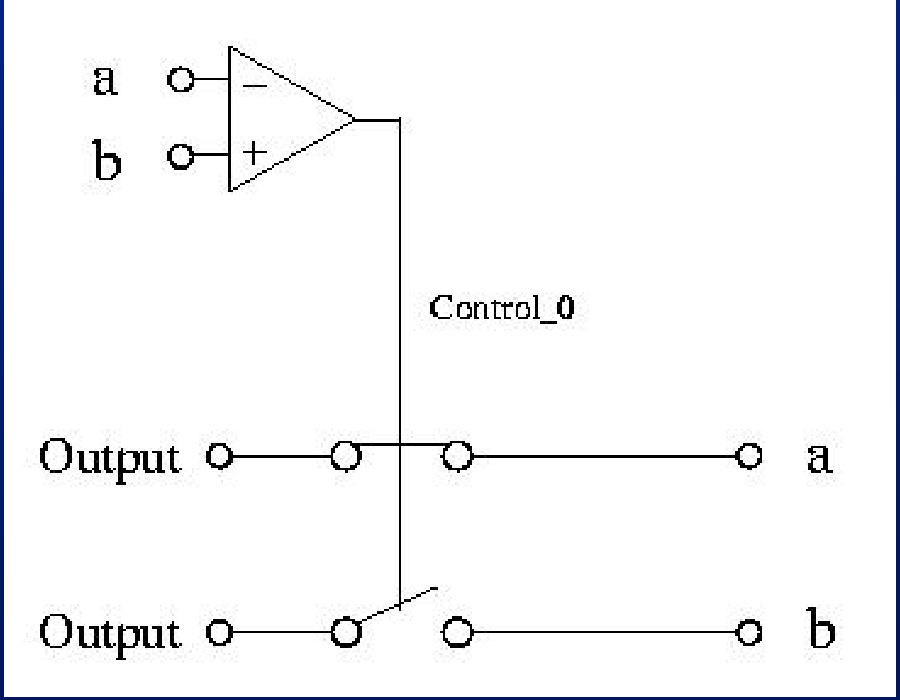
Simultaneous IF statement




```
entity amplifier is
   generic(gain:real := 150.0);
   port(quantity Vin: voltage;
          quantity Vout: out voltage);
end entity amplifier;
--
architecture DC of amplifier is
begin
   if Vin < -15.0/gain use
         Vout == -15.0;
   elsif Vin >15.0/gain use
         Vout == 15.0;
   else
         Vout == gain*Vin;
   end if;
end architecture DC;
```

NEUSYS treatment of simultaneous IF statement

```
if a < b use
        output == a;
else
        output == b;
end if;
```



Example: discretizer synthesis

```
if input(1)>0.6 use
      if input(1)>1.0 use
              output==1.0;
      else output==0.6;
      end use;
elsif input(1)>0.4 use
     output==0.4;
else
    if input(1)>0.2 use
              output==0.2;
   else    output==0.0;
  end use;
end use;
```

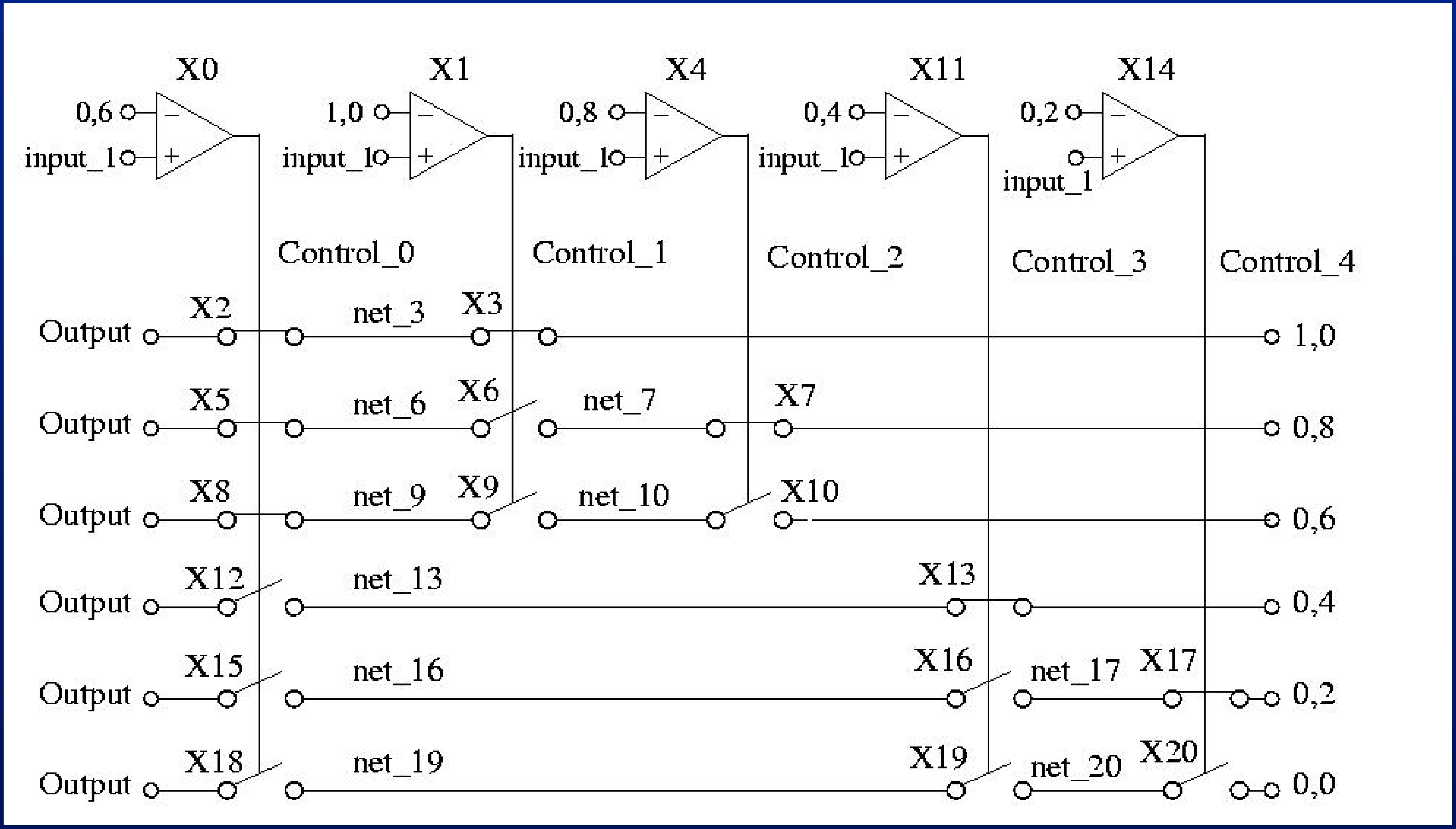
1. Outermost **if** creates a comparator that drives a control signal for different switches:
   X0 input_1 0.6 Control_0 Vref Comparator

2. Next **if**, nested in first one, translates as:
   X1 input_1 1.0 Control_1 Vref Comparator

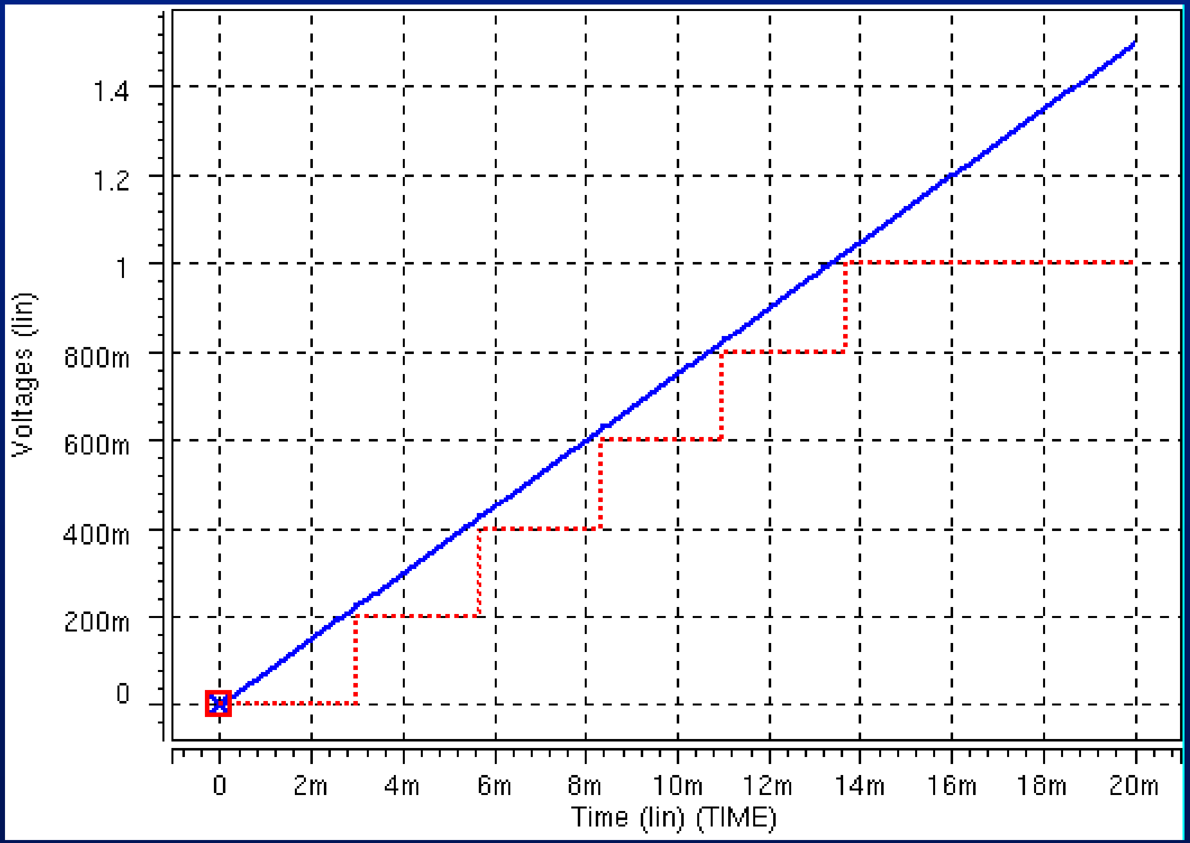
3. Then, first *simultaneous statement* (output == 1.0) is processed to produce:
   X2 output net_3 Control_0 Vdd Vss Switch_1
   X3 net_3 1.0 Control_1 Vdd Vss Switch_1

etc.

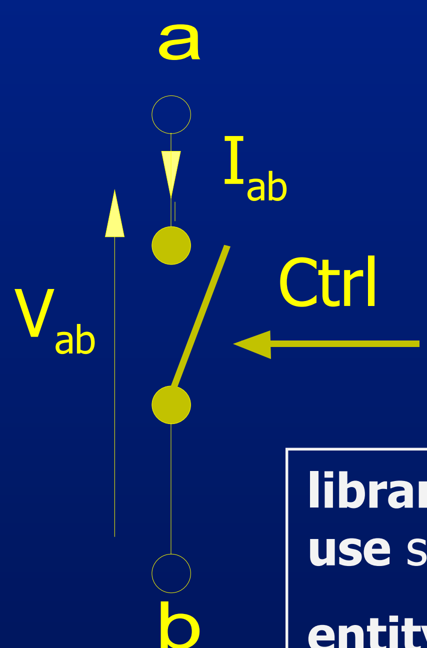
Complete synthesised netlist for discretizer



HSPICE simulation of synthesised netlist



Simultaneous CASE statement



std_logic values:
'U' -- Uninitialized
'X' -- Forcing Unknown
'0' -- Forcing 0
'1' -- Forcing 1
'Z' -- High Impedance
'W' -- Weak Unknown
'L' -- Weak 0
'H' -- Weak 1
'-' -- Don't care

```
library std_logic_1164;
use std_logic_1164.all;

entity IdealSwitch is
        port( Ctrl: std_logic := 'U';
              terminal a,b: electrical);
end entity;
```

Department of Electronics and Computer Science
Southampton University

Simultaneous PROCEDURAL statement

Shichman-Hodges MOS model:

$$I_d = KSV_{ds}(2V_{gst} - V_{ds}); \quad V_{gst} \geq V_{ds}$$

$$I_d = KSV_{gst}^2(1 + \lambda V_{ds}); \quad V_{gst} < V_{ds}$$

$$V_{gst} = V_{gs} - V_t$$

$$S = \frac{W}{L}$$

```
entity MOS is
  generic( K: real := 2.0E-4; --A/V^2
           W: real := 3.0E-6;
           L:  real := 6.0E-6;
           Vt: voltage := 1.0;
           lambda: real := 0.02);
  port( terminal drain,
                 gate,
                 source: electrical);
end entity;
```

Department of Electronics and Computer Science
Southampton University

Simultaneous PROCEDURAL statement

```
architecture Shichman-Hodges of
        MOS is
  quantity Vgs across
                   gate to source;
  quantity Vds across
         Ids through
                   drain to source;

begin
  procedural is -- eqn. for Ids

    ...

  end procedural;
end architecture;
```

```
procedural is -- eqn. for Ids
    variable KS: real;
    variable Vgst: voltage;
begin
  -- calculate model parameters:
  KS := K*W/L;
  Vgst := Vgs-Vt;
  -- specify equation for Ids:
  if Vgst <=0.0 then
    Ids:=0.0;
  elsif Vgst < Vds then
    Ids:=KS*Vgst**2*(1.0-lambda*Vds);
  else
    Ids:=KS*Vds*(2.0*Vgst-Vds);
  end if;

end procedural;
```

Department of Electronics and Computer Science
Southampton University

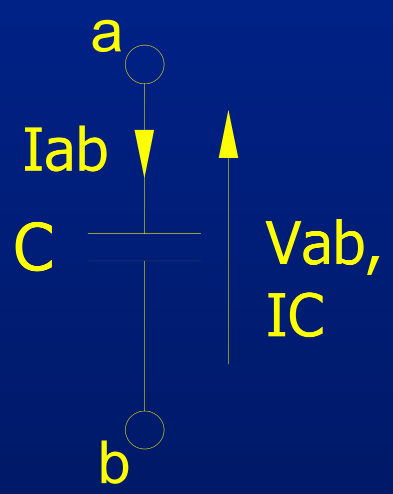
BREAK statement

- A **break** statement overrides quantity values
- It executes concurrently with the analogue model
- For any concurrent break statement there is an associated process
- **break** statement can be synthesised

Department of Electronics and Computer Science
Southampton University

Concurrent break statement

Setting initial conditions

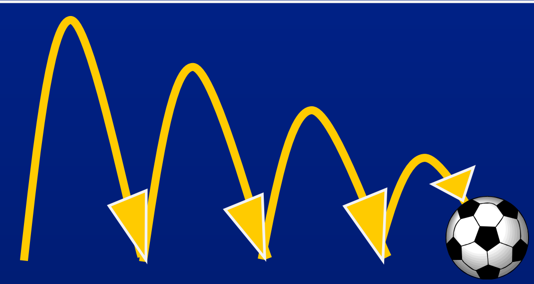



```
architecture DiffEqn of capacitor is
  quantity Vab across Iab through a to b;
begin

 -- initial condition
  break Vab => IC;

  -- differential equation
  Iab == C*Vab'DOT;
end architecture DiffEqn;
```

```
entity capacitor is
  generic(C:real := 0.0;
          IC: voltage := 0.0);
  port(terminal a,b: electrical);
end entity capacitor;
```

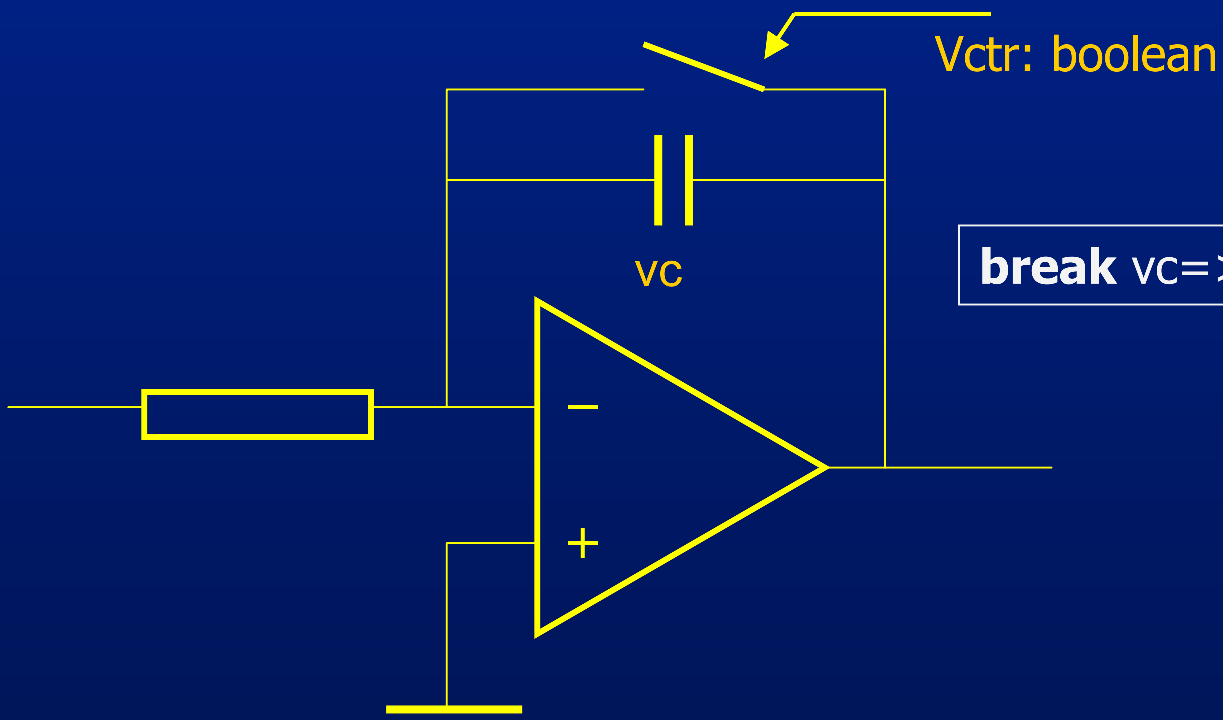
Concurrent break statement

Modelling of waveform discontinuities



```
architecture bouncing of ball is
  quantity v: velocity  := 0.0;
  quantity s: displacement  := 10.0;
  constant G: real  := 9.81; -- G-force
  constant AirResistance: real  := 0.1;
begin
   s'DOT == v;
   if v > 0.0 use
            v'DOT == -G  - AirResistance*v**2; -- falling
   else   v'DOT == -G + AirResistance*v**2; -- rising
  end if;
  -- introduce discontinuity when ball hits ground:
  break v => -v when v'ABOVE(0.0) and not s'ABOVE(0.0);
end architecture bouncing;
```

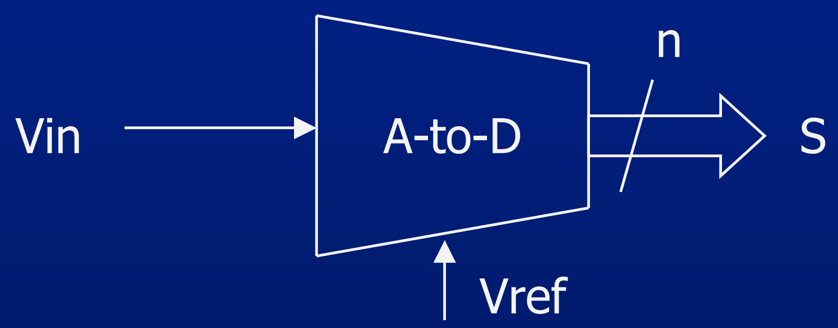
Concurrent **break** can be synthesised



break vc=>0.0 **when** Vctr;

A/D interfacing with Q'ABOVE attribute

A/D tracking converter




```
entity AD is
   generic(Vref: voltage := 2.56;          -- reference voltage
                      n: positive := 16);         -- n bits
   port (quantity Vin: in voltage;         -- analogue input
                s: inout bit_vector(1 to n)); -- digital output
end entity;
```

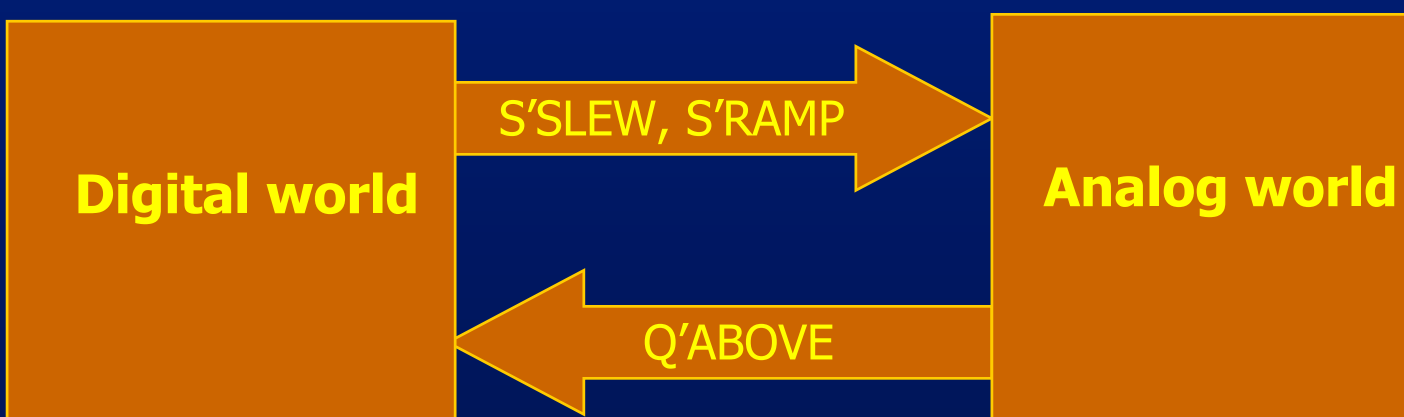
A/D interfacing using Q'ABOVE
A/D tracking converter

```
architecture tracking of AD is
  -- declare a free local quantity to monitor the tracking error
  quantity Error: voltage := 0.0;
begin
   Error == Vin - s'RAMP*Vref/n;  -- error equation
   --tracking
   s <=      s-1  when Error'ABOVE(Vref/n)
       else  s+1 when not Error'ABOVE(0.0);
end architecture tracking;
```

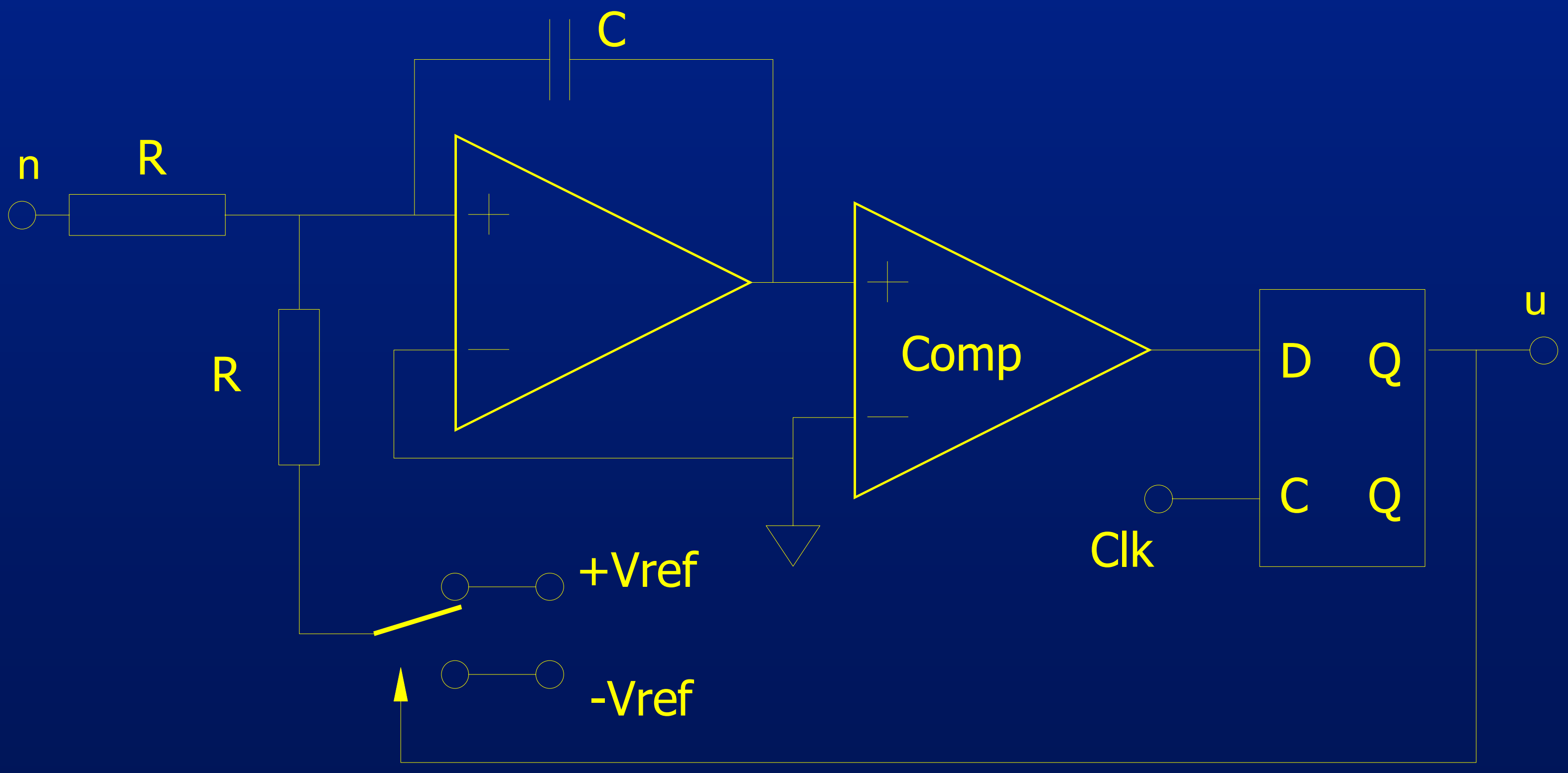
Department of Electronics and Computer Science
Southampton University

A/D and D/A interfacing summary

- Q'ABOVE(E) is a signal that announces discrete events in response to quantity variations.
- S'RAMP and S'SLEW are quantities that announce analog variations in response to discrete events on signals.



Mixed-signal behaviour – Sigma-Delta modulator



Sigma-Delta modulator
entity declaration

```
entity SigmaDelta is
  generic (R: real := 1.0E3;
               C:real := 100.0E-9;
               Vref: voltage := 1.5);
 port (terminal n: electrical;
          Q: inout bit );
end entity;
```

Sigma-Delta modulator
architecture declaration

```
architecture sampling of SigmaDelta is
  quantity Vin across  Iin through n to  ground;
  quantity Iint: current;
begin
  -- input branch Ohm's Law
  Iin == V/R;

 -- switch & integrator current
  if Q==‘1’ use
        Iint == Iin+Vref/R;
  else Iint == Iin-Vref/R;

  -- integrator
  Vc == C*Iint'INTEG;
...
```

```
 ...

  --D-type flip-flop model
  DFF: process (Clk) is
  begin
    if Clk'EVENT and Clk=‘1’ then
      -- compare Vc with 0 and set Q
       if Vc > 0.0 then
              Q <=  '1';
      else Q <=  '0';
      end if;
    end if;
   end process;
end architecture switching;
```

Department of Electronics and Computer Science
Southampton University

Voltage-controlled oscillator (VCO) with phase integration

Vin

$$\varphi(t) = 2\pi \int_{\tau=0}^{t} \max\{f_{\min}, f_c + [V_{in}(\tau) - V_c]\Delta f\}\, d\tau$$

$$V_{out} = V_M \sin[\varphi(t)]$$

Vout

Department of Electronics and Computer Science
Southampton University

VCO waveforms

VCO with phase integration
entity declaration

```
entity VCO is
  generic ( f0: real := 1.0E6;    -- VCO frequency at Vin = 0
            df: real := 0.5E6   -- [Hz/V], frequency characteristic slope
          );
  port( quantity Vin: in voltage;
        terminal OutT: electrical);
end entity VCO;
```

VCO with phase integration
architecture declaration

```
architecture PhaseIntegrator of VCO is
    quantity phase : real := 0.0;
    quantity Vout across Iout through OutT to ground;
begin
  -- initial condition
        break phase => 0.0;
  -- keep phase within 0.. 2pi
        break phase => phase mod math_two_pi
                                     when Phase'ABOVE(math_two_pi);
  -- phase equation
        phase'DOT == math_two_pi*max(0.5E6, f0+Vin*df);
  -- output voltage source equation
        Vout == sin(phase);
end architecture PhaseIntegrator;
```

Frequency domain support in VHDL-AMS

- The frequency domain model (AC model) is obtained from the linearized time-domain model

$$Q'DOT \Rightarrow j\omega \cdot Q(j\omega)$$

$$Q'INTEG \Rightarrow \frac{Q(j\omega)}{j\omega}$$

$$Q'DELAYED(T) \Rightarrow Q(j\omega) \cdot e^{j\omega T}$$

$$\omega = 2\pi \cdot FREQUENCY$$

Department of Electronics and Computer Science
Southampton University

Frequency domain support in VHDL-AMS (cont)

- Source quantities provided to specify AC sources

quantity V: voltage **spectrum** mag,phase;
quantity Inoise: current **noise** sqrt(4.0*TEMP0*Boltzman/R);

- Laplace (s-domain) and z-domain transfer functions supported via quantity attributes e.g. Q'LTF, Q'ZTF

Q'LTF is recognised by FIST in filter synthesis

Department of Electronics and Computer Science
Southampton University

LTF attribute in s-domain modelling

Use of built-in attribute LTF: Q’LTF(a,b)

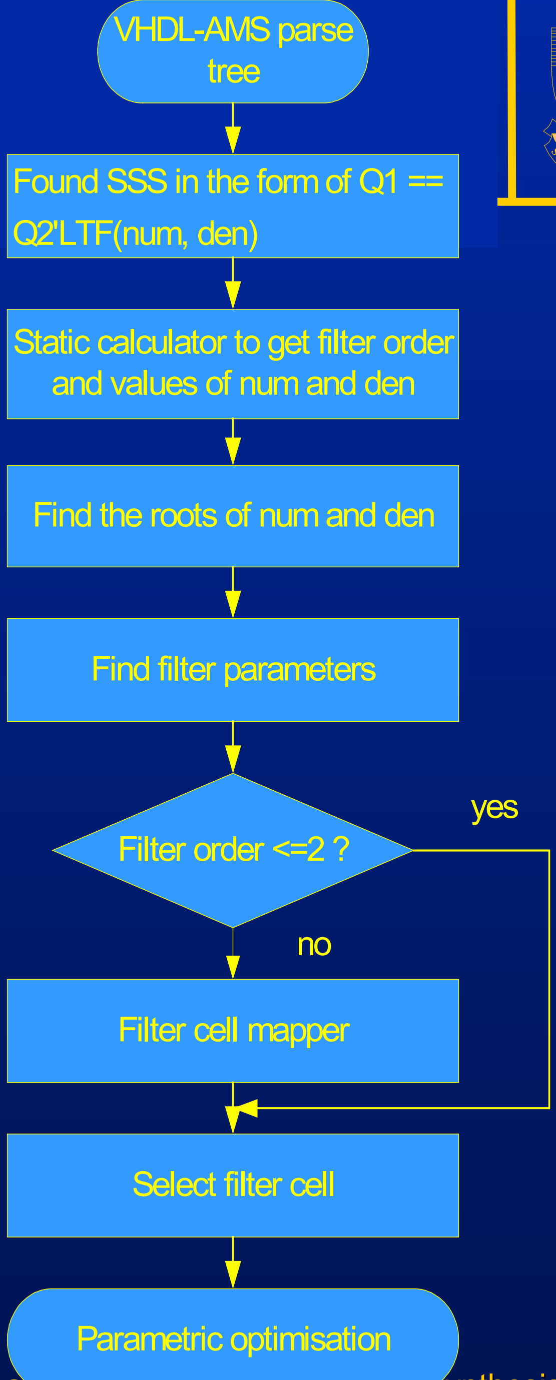
Vin(s)

$$\frac{a_n s^n + \ldots + a_1 s^1 + a_0}{b_m s^m + \ldots + b_1 s^1 + b_0}$$

Vout(s)

s-domain filter model

```
entity Filter is
   generic (N : positive := 1; -- numerator order
               M : positive := 1;  -- denominator order
               A: real_vector(1 to N) := (1=>1.0); -- num coefficients
               B: real_vector(1 to M) := (1=>1.0));-- den coefficients
   port (terminal InT,OutT: electrical);
end entity;

architecture behaviour of Filter is
  quantity Vin across InT to ground;
  quantity Vout across Iout through OutT to ground;
begin

     Vout == Vin'LTF(A,B);

end architecture behaviour;
```



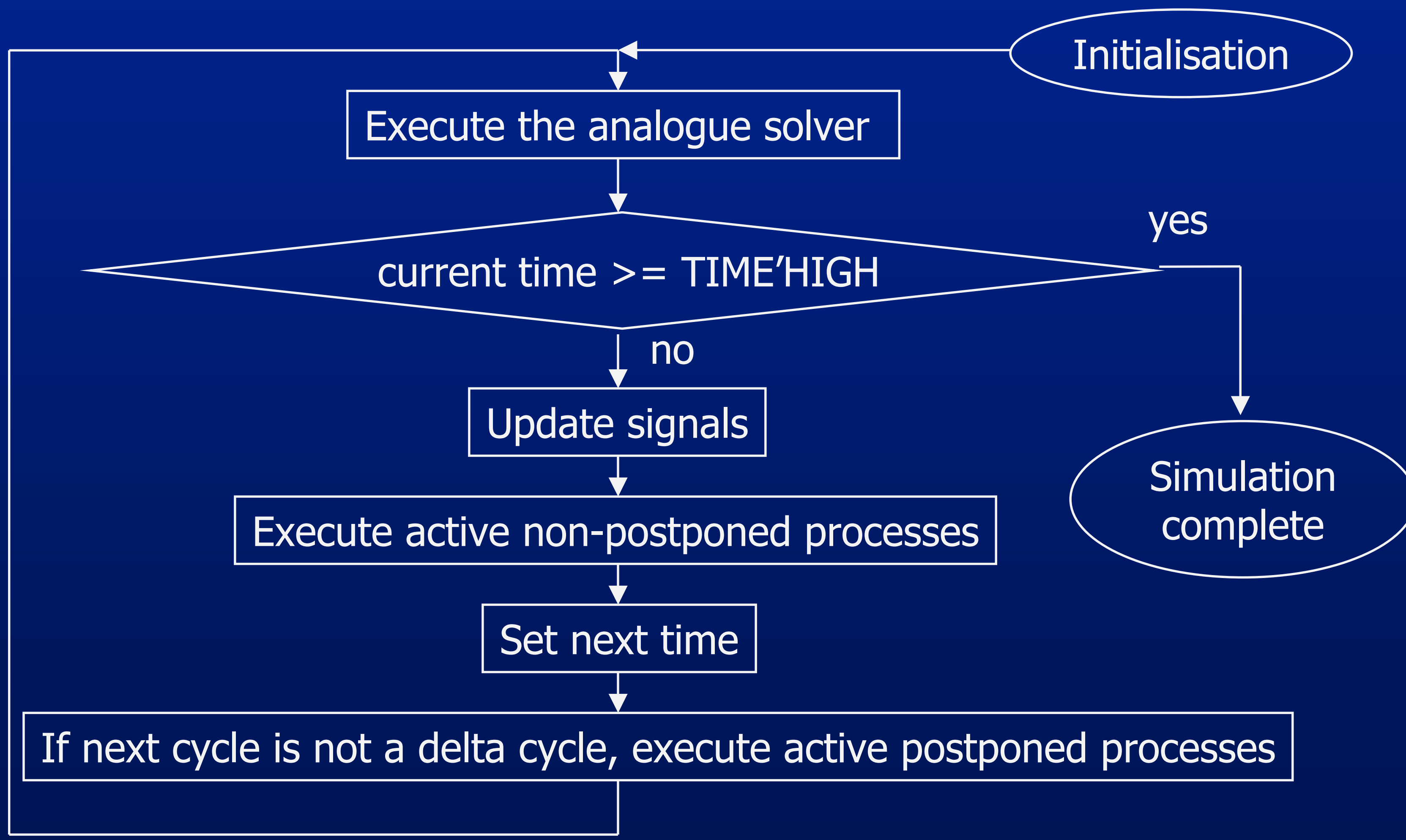
FIST synthesis algorithm for LTF constructs

VHDL-AMS execution kernel

- The kernel process:
  - invokes the analogue solver to determine the values of quantities within the model
  - executes the user processes thus causing the values of signals to be updated
- If the model contains no quantities, the analogue solver is not invoked
- The kernel process contains a driver for the new predefined signal DOMAIN

Department of Electronics and Computer Science
Southampton University

VHDL-AMS simulation cycle



Introduction to VHDL-AMS concluding remarks

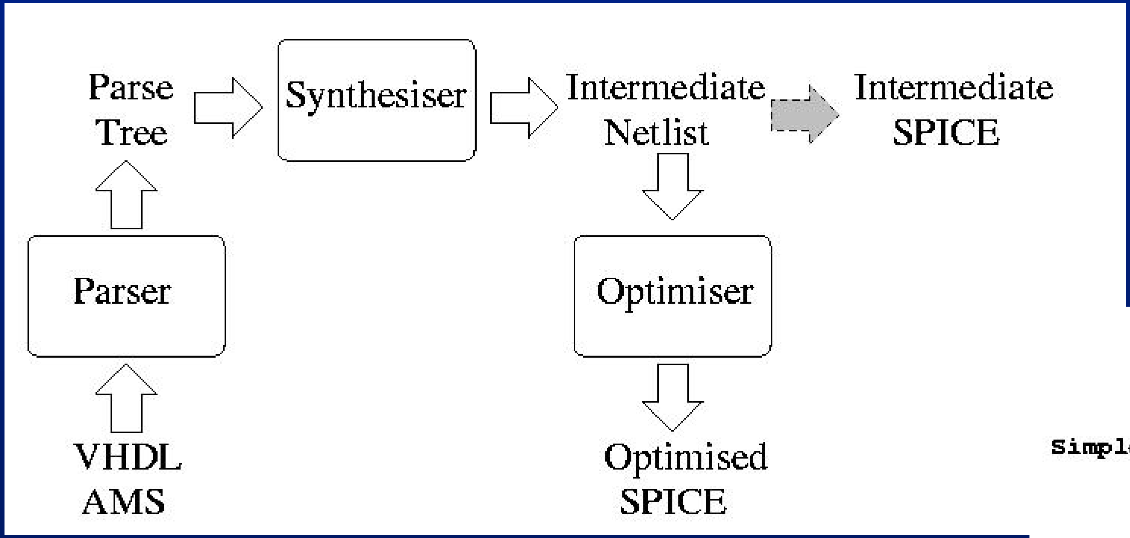
- VHDL-AMS is a strict superset of VHDL extended to allow modelling of continuous dynamic systems in addition to VHDL discrete system models.
- VHDL modelling power extended to non-electrical domains.
- VHDL-AMS is likely to become a major description tool for analogue and mixed-signal synthesis.

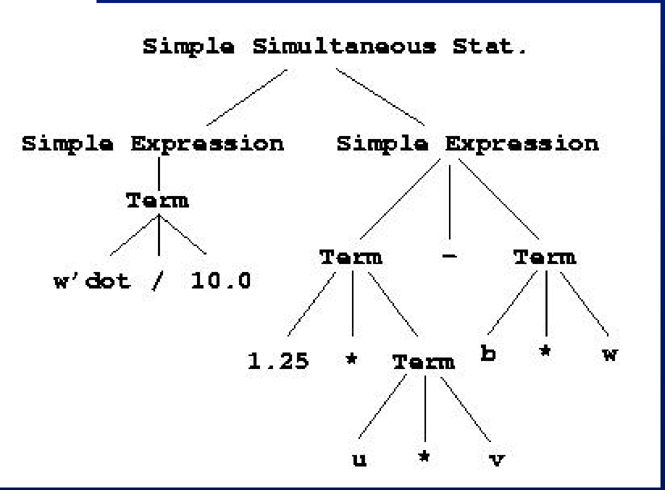
Department of Electronics and Computer Science
Southampton University

NEUSYS - an early VHDL-AMS based system

- Silicon implementations of ANNs, especially large analogue networks of the type required in fuzzy logic and control systems, might provide an important area for analogue synthesis applications.
  - Biologically inspired ANNs are often used in robotic arm control systems as they reflect the basic structure and behaviour of the human spinal cord, focusing on the motor control of the superior limbs.
- NEUSYS - an automated synthesis system that converts VHDL-AMS descriptions of highly interconnected networks of neural cells into HSPICE net lists.

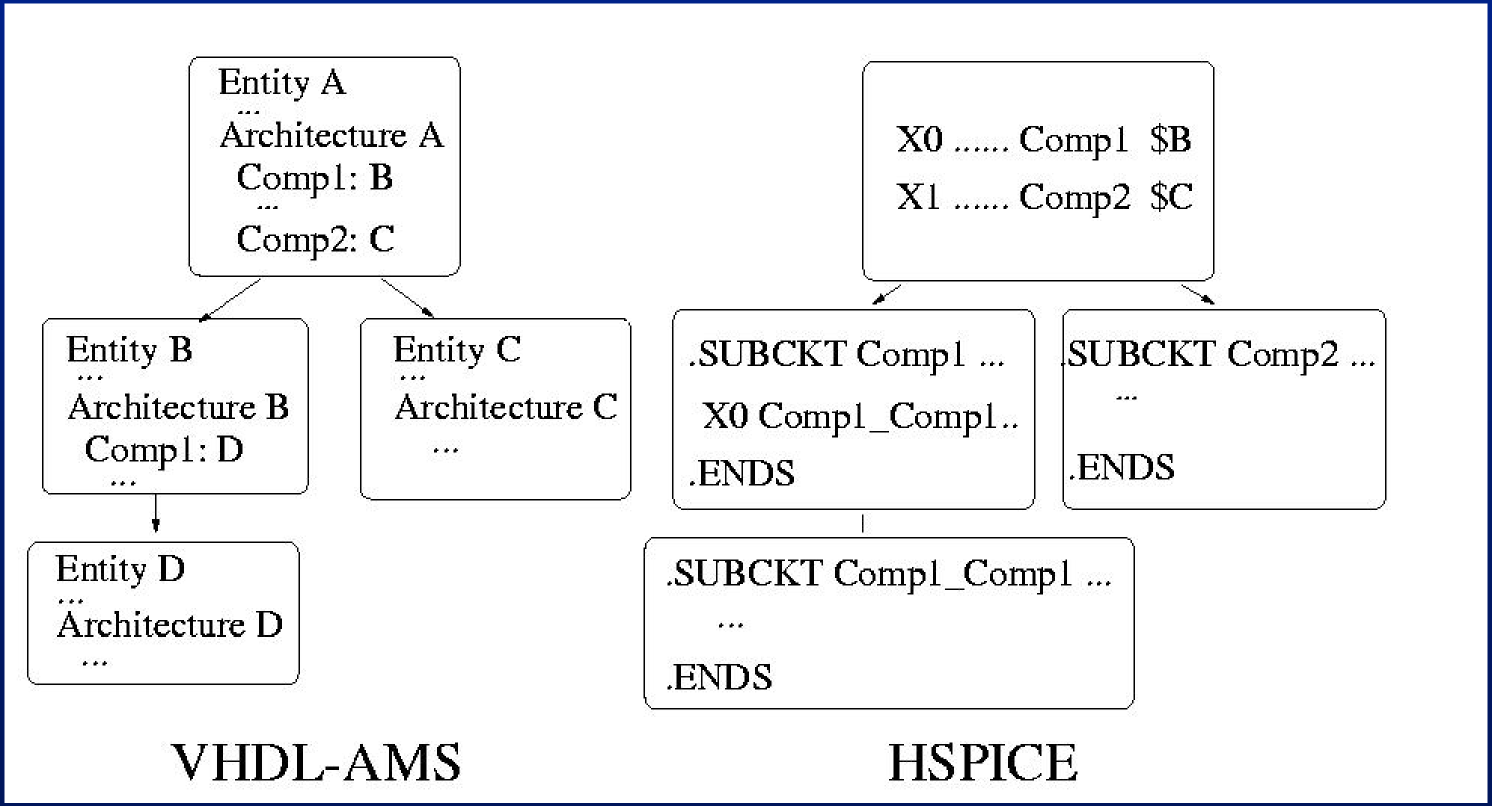
NEUSYS structure





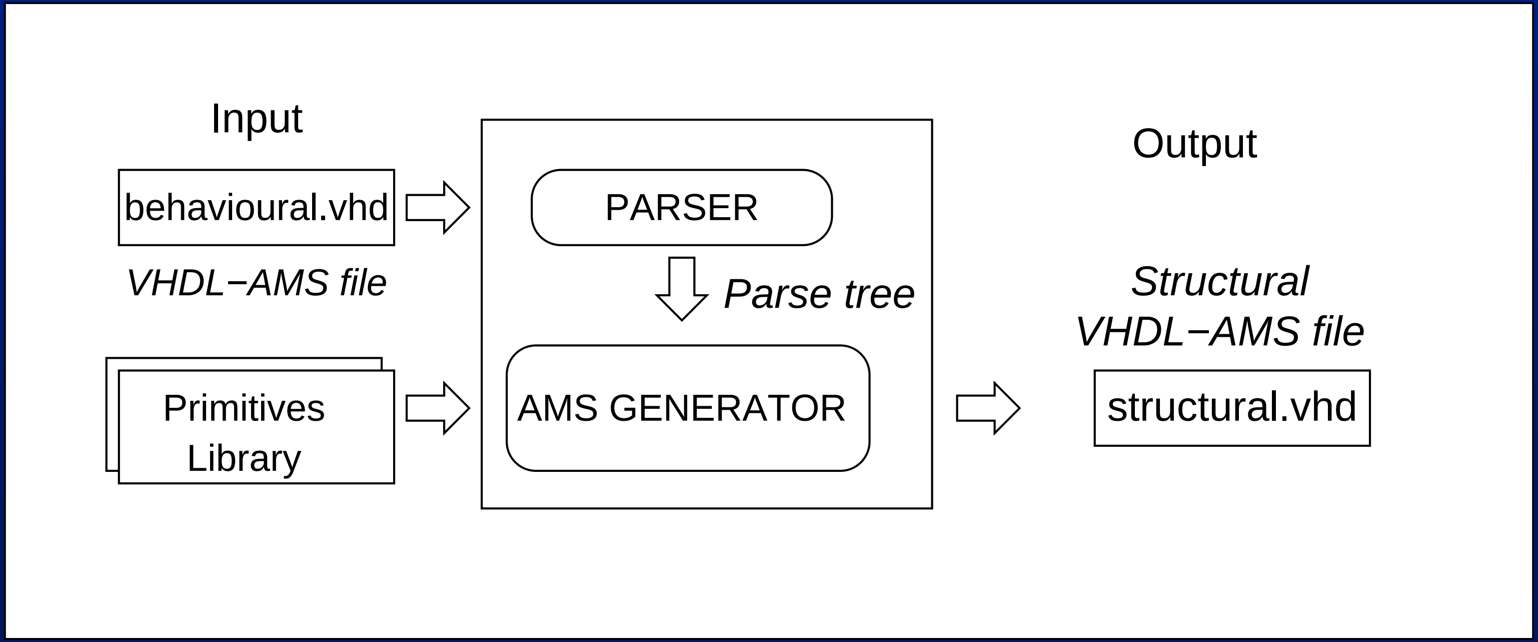
Sample parse tree

NEUSYS HSPICE output



Department of Electronics and Computer Science
Southampton University

NEUSYS structural VHDL-AMS output



Department of Electronics and Computer Science
Southampton University

Recursive translation in NEUSYS to support VHDL-AMS hierarchy

(VHDL_AMS component to SPICE .SUBCKT)

VHDL-AMS

Architecture...
begin
Component Instantiation: Entity
port map (formal=>actual)
...

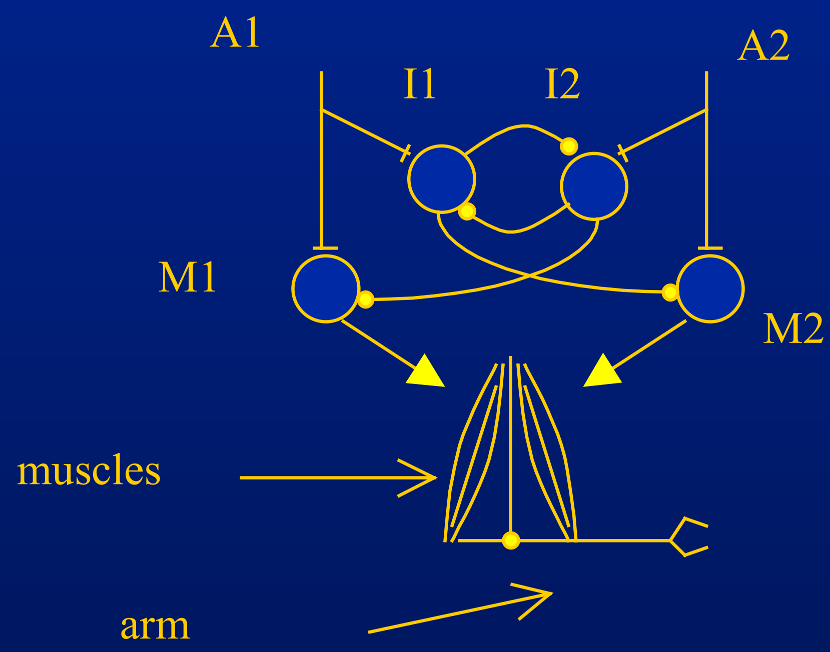
HSPICE

...
X1 *actual* Component Instantiation
...
.SUBCKT *formal* Comp. Instant.
subcircuit body
.ENDS

Department of Electronics and Computer Science
Southampton University

ANN synthesis example

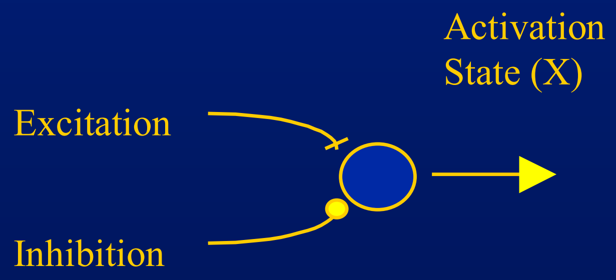
A neural system controlling a pair of antagonistic muscles



A1,A2 - excitations

M1,M2 - motor neurons

I1,I2 alpha inter neurons



Department of Electronics and Computer Science
Southampton University

ANN muscle control system synthesis

Outline of VHDL-AMS model: inter alpha neuron and motor neuron

```
architecture eqn of Inter_neuron is
begin
  -- initial condition
  break IaI => 2.0;
  -- behavioural equation:
  IaI'dot == (10.0-IaI)*A-(IaI+1.0)*(1.0+IaI_opp);
end architecture eqn;
```

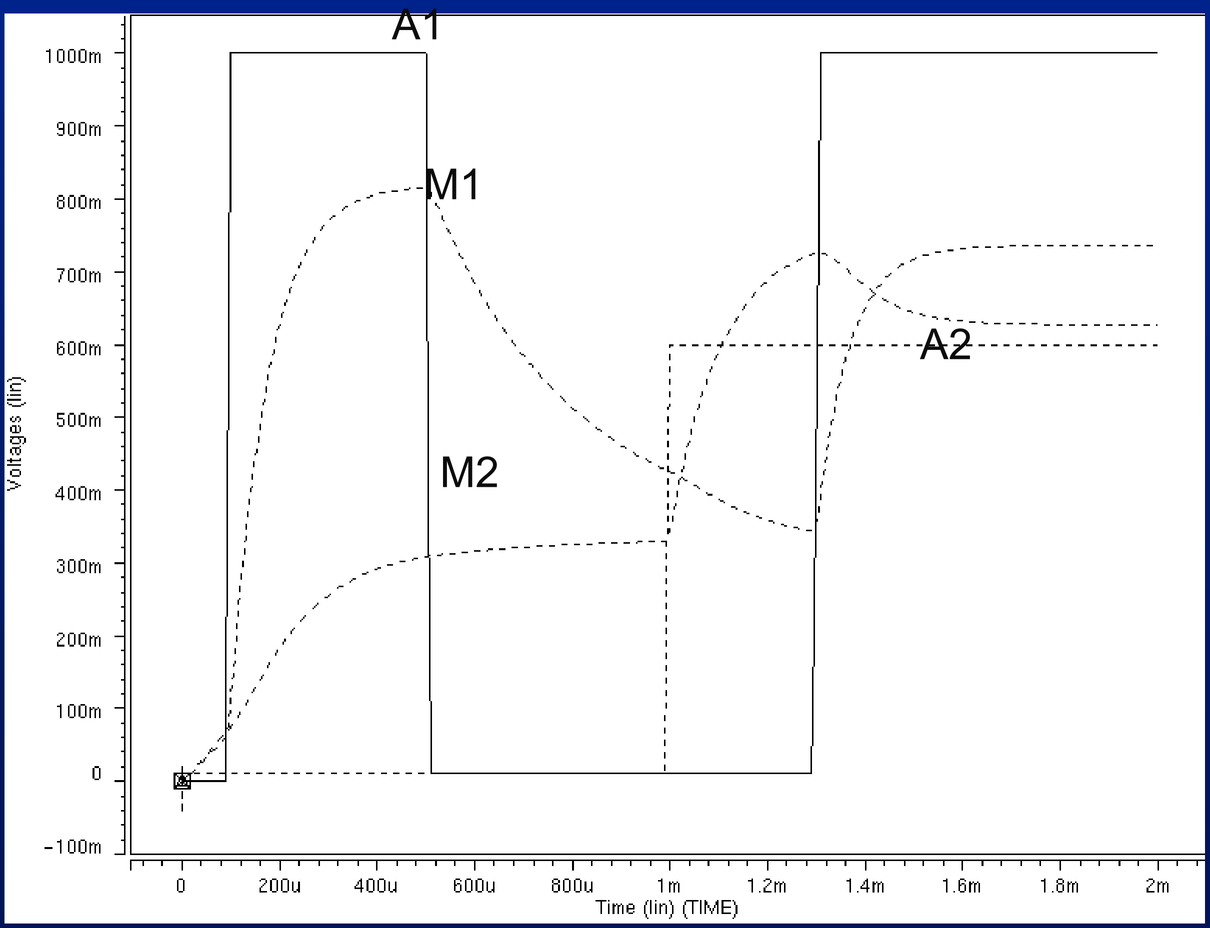
```
architecture eqn of Motor_neuron is
  quantity lambda:real:=3.0;
begin
  -- initial conditions
  break M => 0.0;
 -- behavioural equation:
  M'dot == (lambda-M)*A-M+1.6)*(0.2+IaI_opp);
end architecture eqn;
```

ANN muscle control (cont.)

```
architecture structure of network_1 is
  quantity IaI11,IaI12,IaI21,IaI22: real;
begin
 C_IaI1: Inter_Neuron
   port map (A=>A1,IaI =>IaI12,IaI=>IaI11,IaI_opp
=>IaI21);
 C_IaI2: Inter_Neuron
   port map (A=>A2,IaI=>IaI21,IaI=>IaI22,IaI_opp=>
IaI12);
 C_M1: Motor_Neuron
   port map (A=>A1,IaI_opp=>IaI22, M=>out1);
 C_M2: Motor_Neuron
   port map (A=>A2, IaI_opp=>IaI11, M=>out2);
end architecture structure;
```

Department of Electronics and Computer Science
Southampton University

ANN muscle control - HSPICE simulation



NEUSYS also used successfully to synthesise complex types of ART networks

- ART (Adaptive Resonance Theory) network
  - Orientation system,
  - Winner selector field (also known as F4 layer),
  - Class manager
  - Several monochannel sub-blocks named F1, F2, F3 and F5 layers.

Details: J. Lopez, G. Domenech, R.Ruiz, T.J.Kazmierski, “AUTOMATED HIGH LEVEL SYNTHESIS OF HARDWARE BUILDING BLOCKS PRESENT IN ART–BASED NEURAL NETWORKS, FROM VHDL–AMS DESCRIPTIONS”, Proc. ISCAS 2002, Phoenix, AZ

Department of Electronics and Computer Science
Southampton University

Synthesis of a chaotic dynamic system

VHDL-AMS model of Lorenz Chaos,

```
entity LorenzChaos is
  port (quantity u,v,w: out real); -- unknowns u(t),v(t),w(t)
end entity LorenzChaos;

architecture Chaotic of LorenzChaos is
  constant s: real := 10.0;  -- equation parameters
  constant b: real := 0.266;
  constant r: real := 2.8;
begin
  -- initial condition:
    break u=> 0.05, v=> 0.0,  w=> 0.45;
  -- equation set:
    u'dot/10.0 == v-u                         tolerance "voltage";
    v'dot/10.0 == ((r*u)-(0.1*v))-((u*w)*5) tolerance "voltage";
    w'dot/10.0 == 1.25*(u*v) - (b*w)         tolerance "voltage";
end architecture Chaotic;
```

Department of Electronics and Computer Science
Southampton University

Synthesis of a chaotic dynamic system

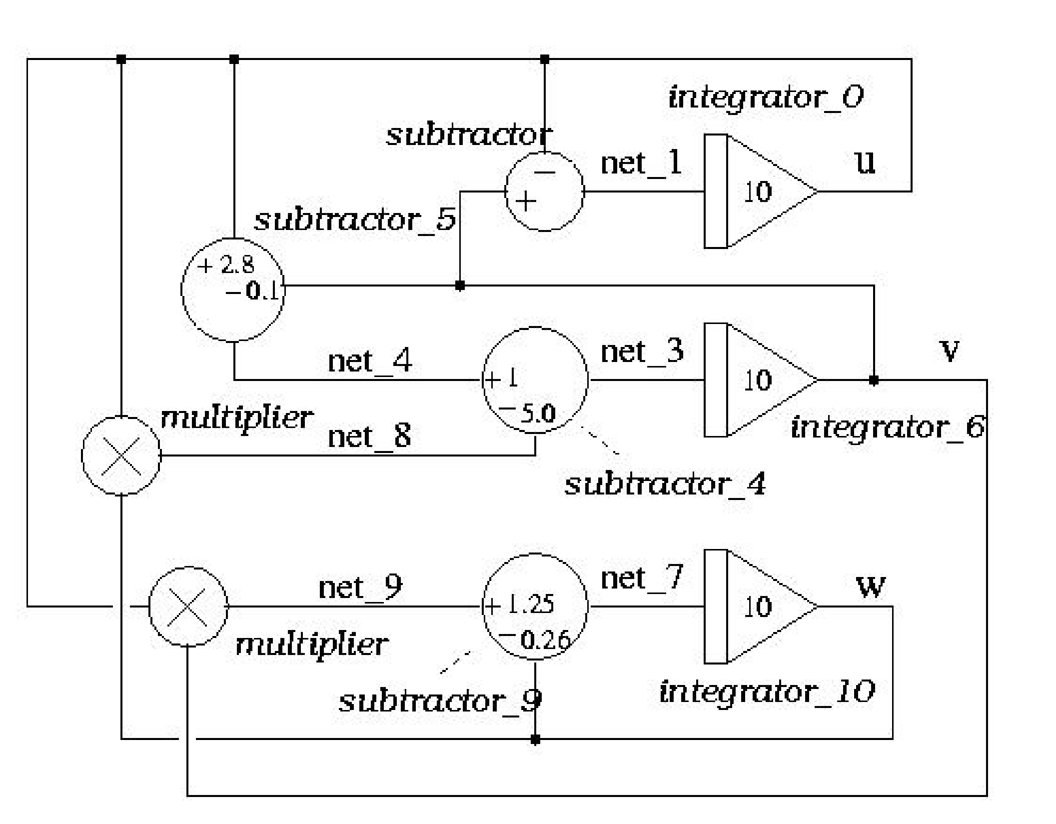
VHDL-AMS model of Lorenz Chaos

NEUSYS intermediate and optimised netlists

```
X0 v u  net_1 Subtractor
X1 net_1  u integrator_0
X2 net_4 net_7  net_3 Subtractor
X3 net_5 net_6  net_4 Subtractor
X4 u  net_5 Gain2.800000
X5 v  net_6 Gain0.100000
X6 net_8  net_7 Gain5.000000
X7 u w  net_8 Multiplier
X8 net_3  v integrator_4
X9 net_11 net_13 net_10 Subtractor
X10 net_12  net_11 Gain1.250000
X11 u v  net_12 Multiplier
X12 w  net_13 Gain0.266000
X13 net_10  w integrator_7
```

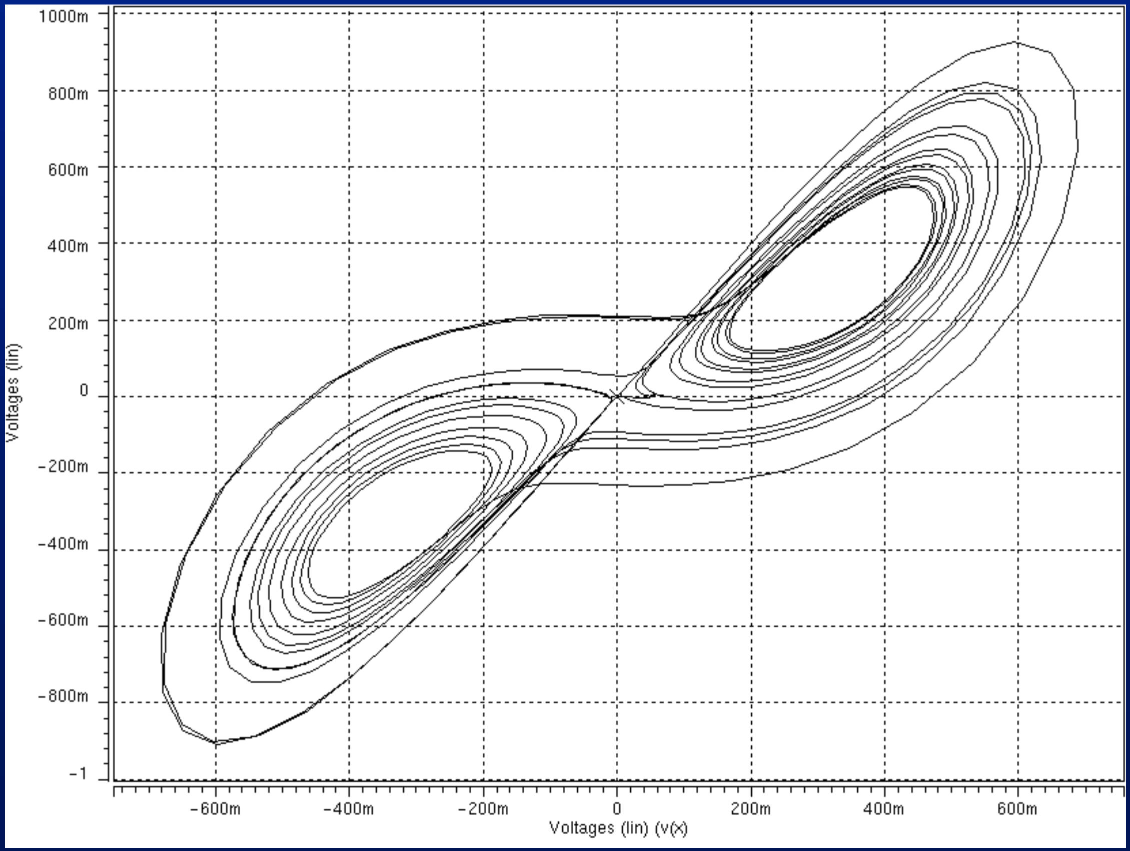
```
X0 v u  net_1 Subtractor
X1 net_1  u integrator_0
X2 net_4 net_8  net_3 Subtractor_4
X3 u v  net_4 Subtractor_5
X4 u w  net_8 Multiplier
X5 net_3  v integrator_6
X6 net_9 w net_7 Subtractor_9
X7 u v  net_9 Multiplier
X8 net_7  w integrator_10
```

Synthesis of Lorenz Chaos,
Opamp level diagram of synthesised circuit



Synthesis of a chaotic dynamic system

Lorenz Chaos,HSPICE simulation of synthesised circuit,



Synthesis of Van der Pol equation

Van der Pol oscillator:

$$\ddot{x} - m(x^2 - 1)\dot{x} + x = 0$$

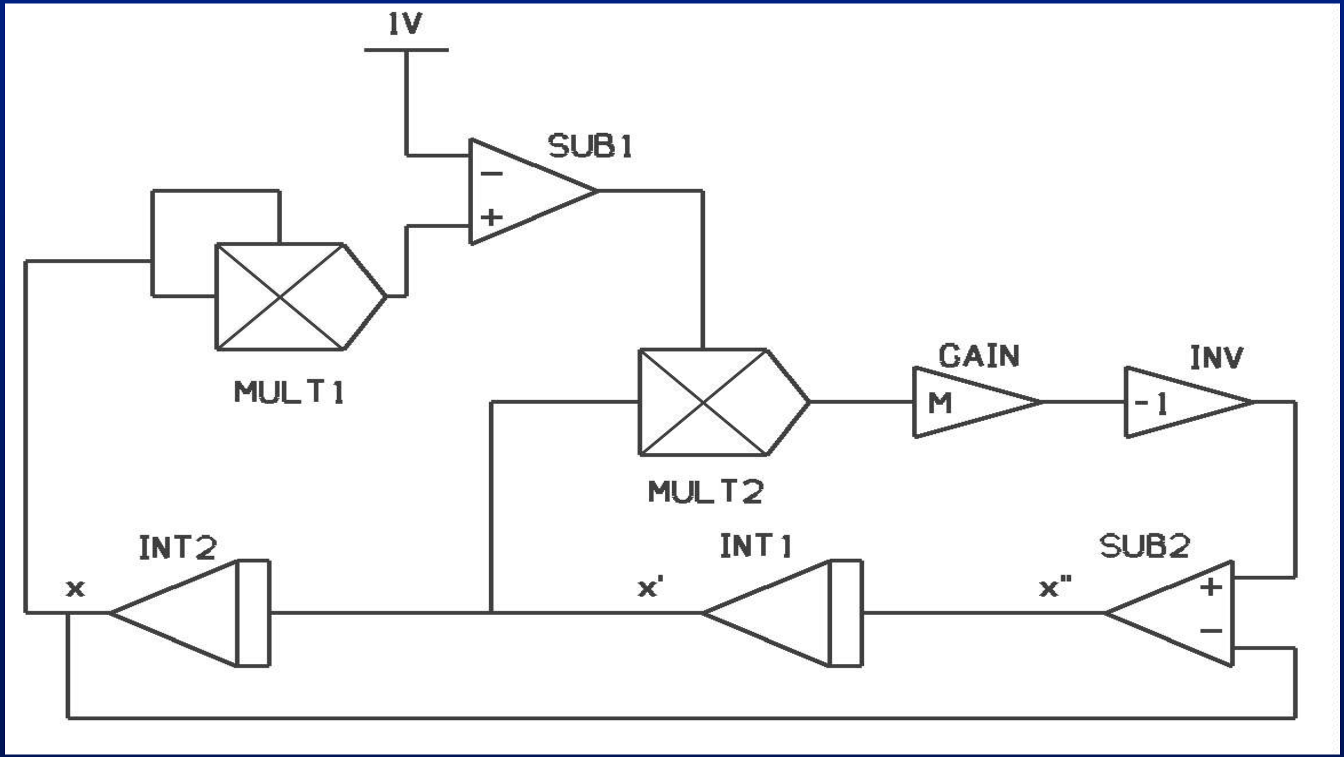
entity VanDerPol **is**
  generic (m: real := 1.0);
  port (**quantity** x: **out** real := 0.1);
end entity;

architecture behaviour **of** VanDerPol **is**
begin
  x == -x'DOT'DOT+m*(1.0-x*x)*x'DOT;
end architecture;

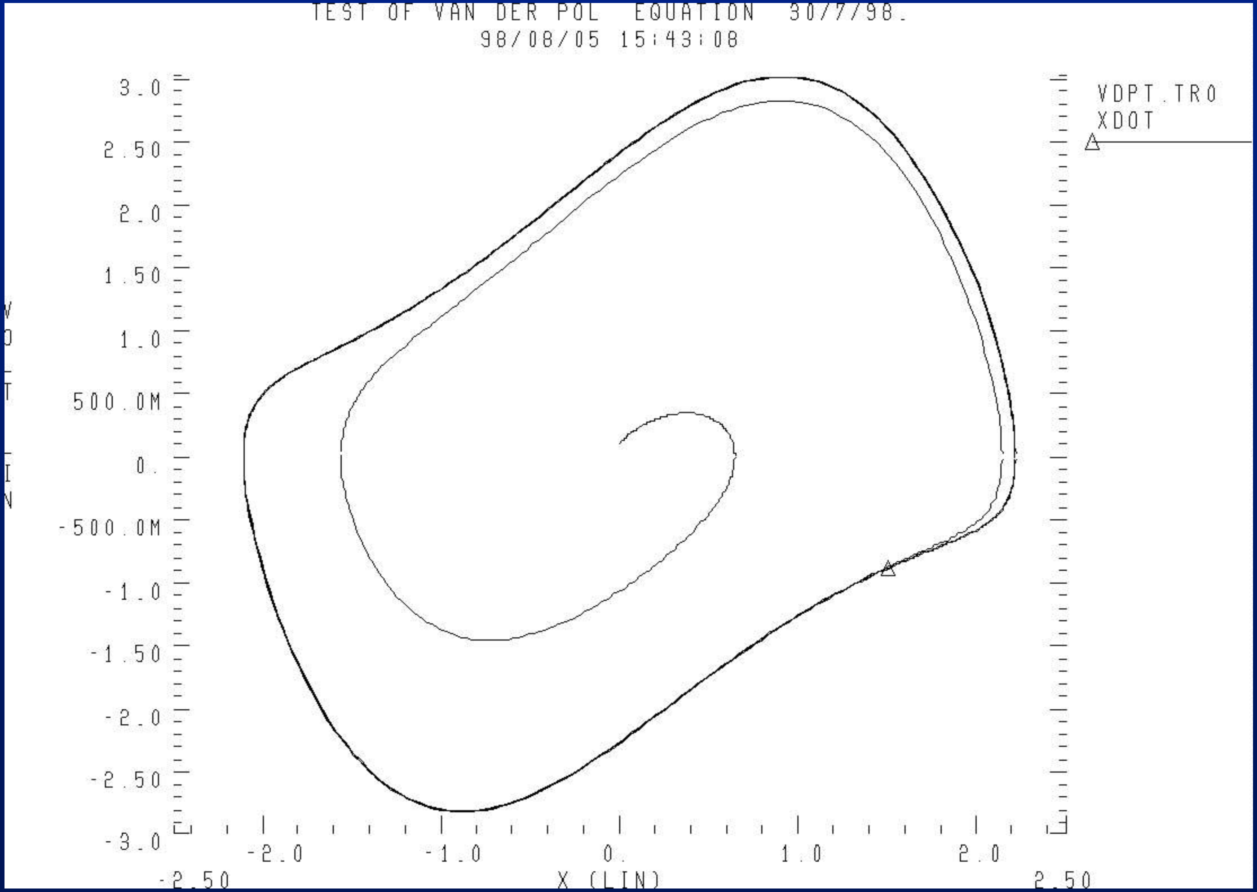
Department of Electronics and Computer Science
Southampton University

Synthesis of Van der Pol equation

opamp-level structure generated by NEUSYS



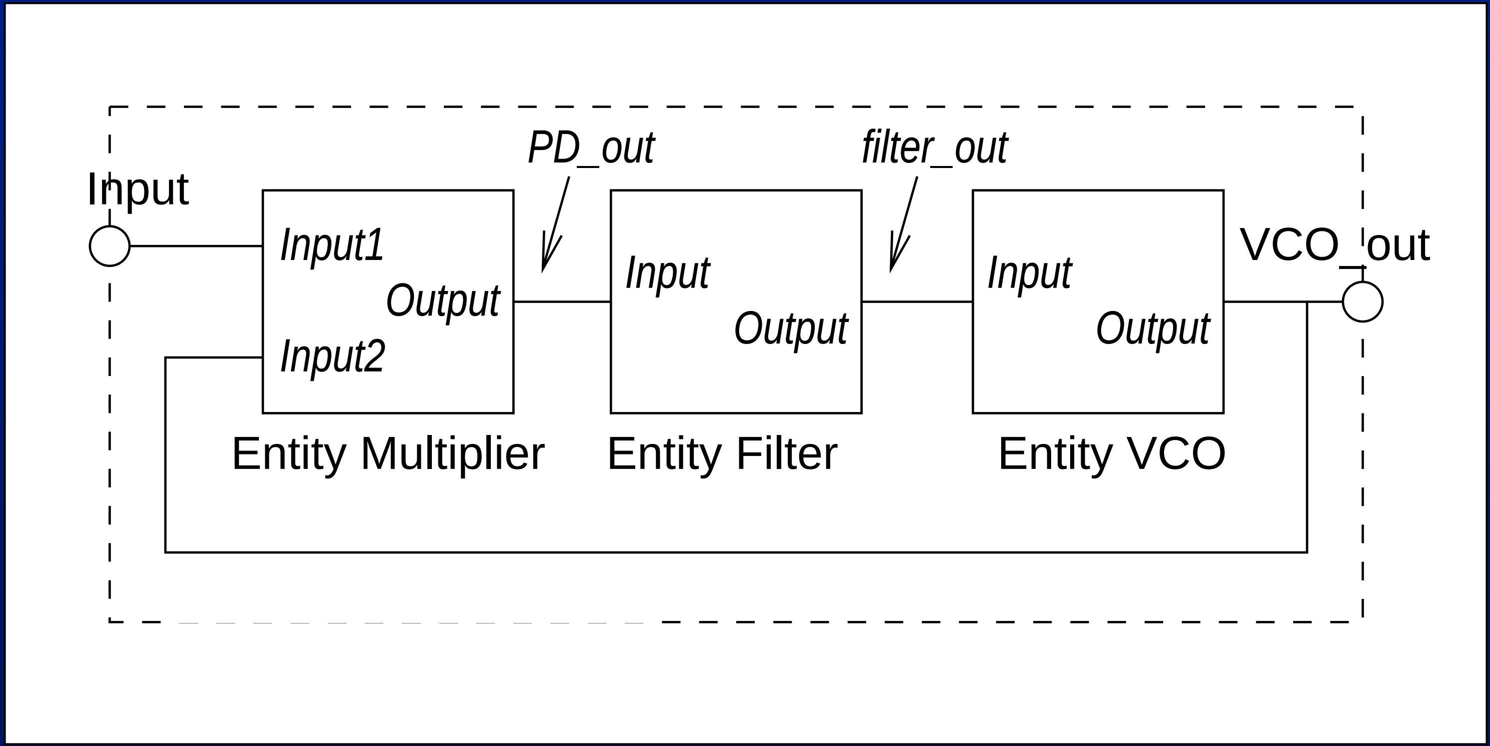
Synthesis of Van der Pol equation
HSPICE simulation



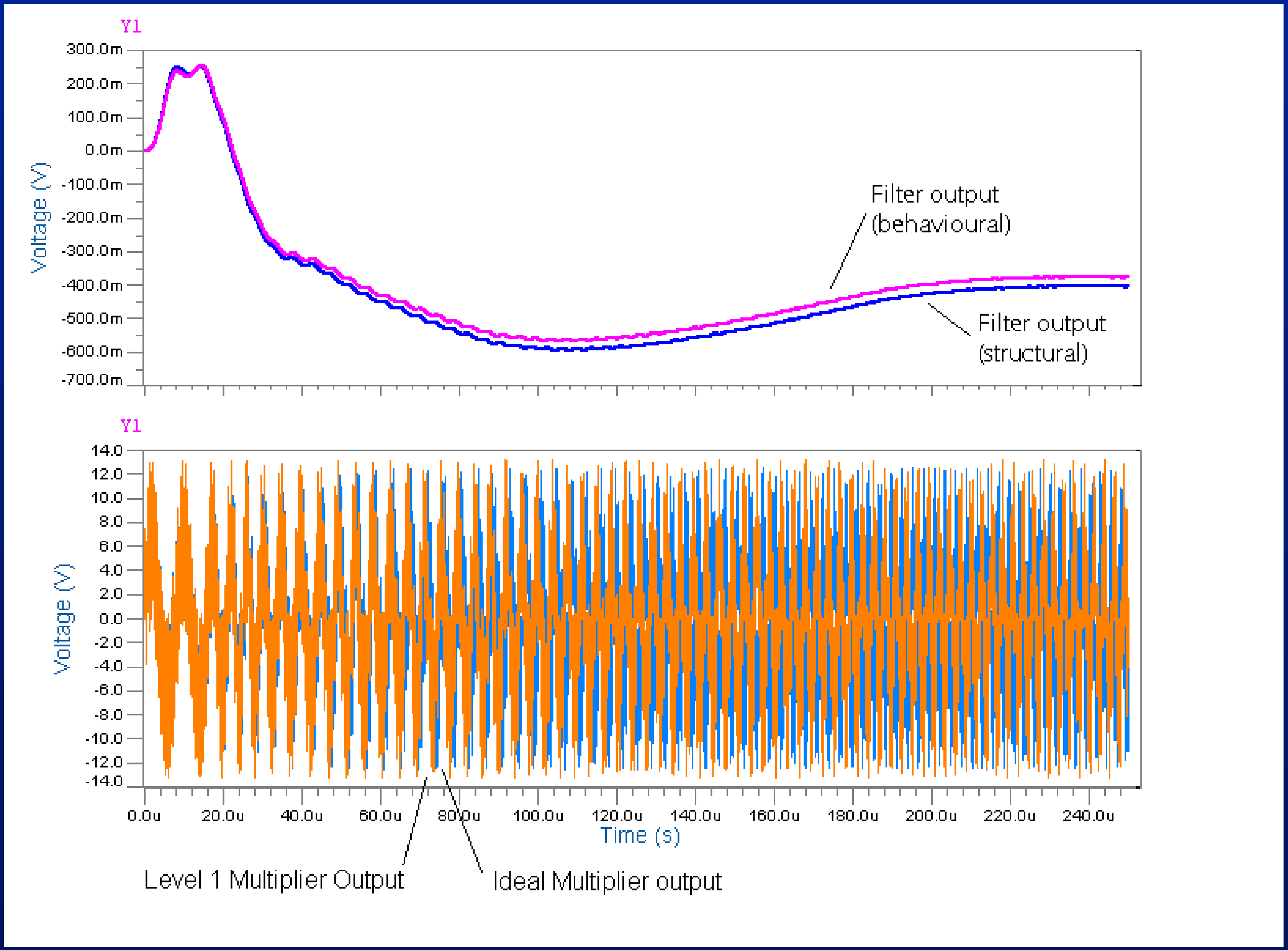
Department of Electronics and Computer Science
Southampton University

NEUSYS structural VHDL-AMS synthesis

Behavioural hierarchical VHDL-AMS to structural VHDL-AMS



Department of Electronics and Computer Science
Southampton University



Accuracy of structural VHDL-AMS synthesis

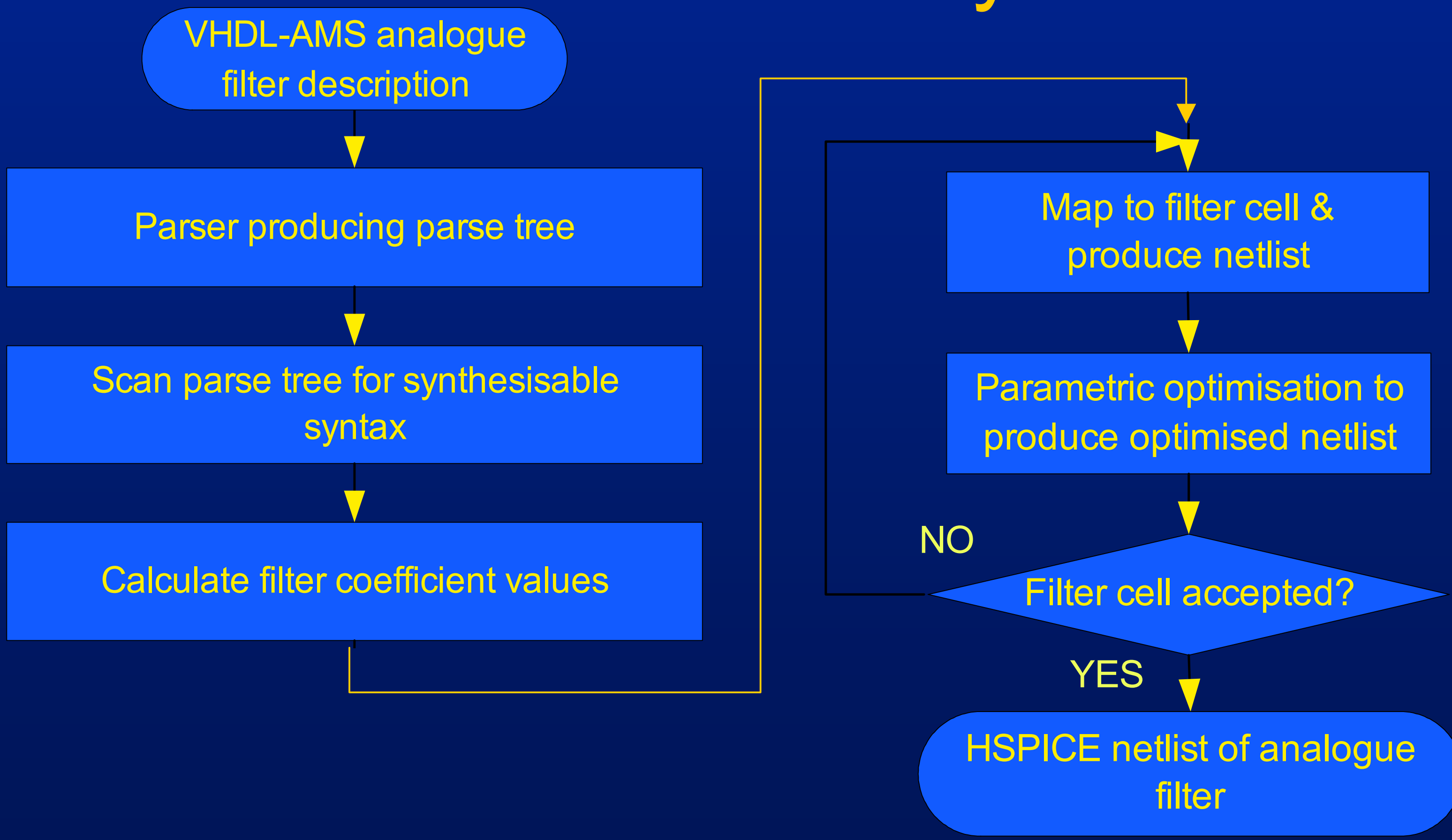
(mostly determined by accuracy of circuit level cells)

FIST – synthesis system for RF filters

- Synthesis of RF filters from high level VHDL-AMS descriptions in time domain or frequency domain.
- FIST recognises synthesisable filter patterns in the VHDL-AMS parse tree .
- Filter candidates are subject to three-tier parametric optimisation comprising:
  - Random search
  - Amoeba search (non-linear simplex)
  - Levenberg-Marquardt optimiser in HSPICE

Department of Electronics and Computer Science
Southampton University

FIST synthesis flow



Sample description of a 1GHz bandpass filter

```
architecture behavioural of filter is
    constant pi: real:=3.142, f: real:=1.0e9;
    constant w: real:= 2.0*pi*f, Q: real: = 1.0;
    constant coeff1: real:= 1.0/(w*w);
    constant coeff2: real:= 1.0/(Q*w);
    constant coeff3: real:= 1.0;
begin
    Vin == coeff1*Vout'dot'dot + coeff2*Vout'dot + coeff3*Vout;
end architecture behavioural;
```

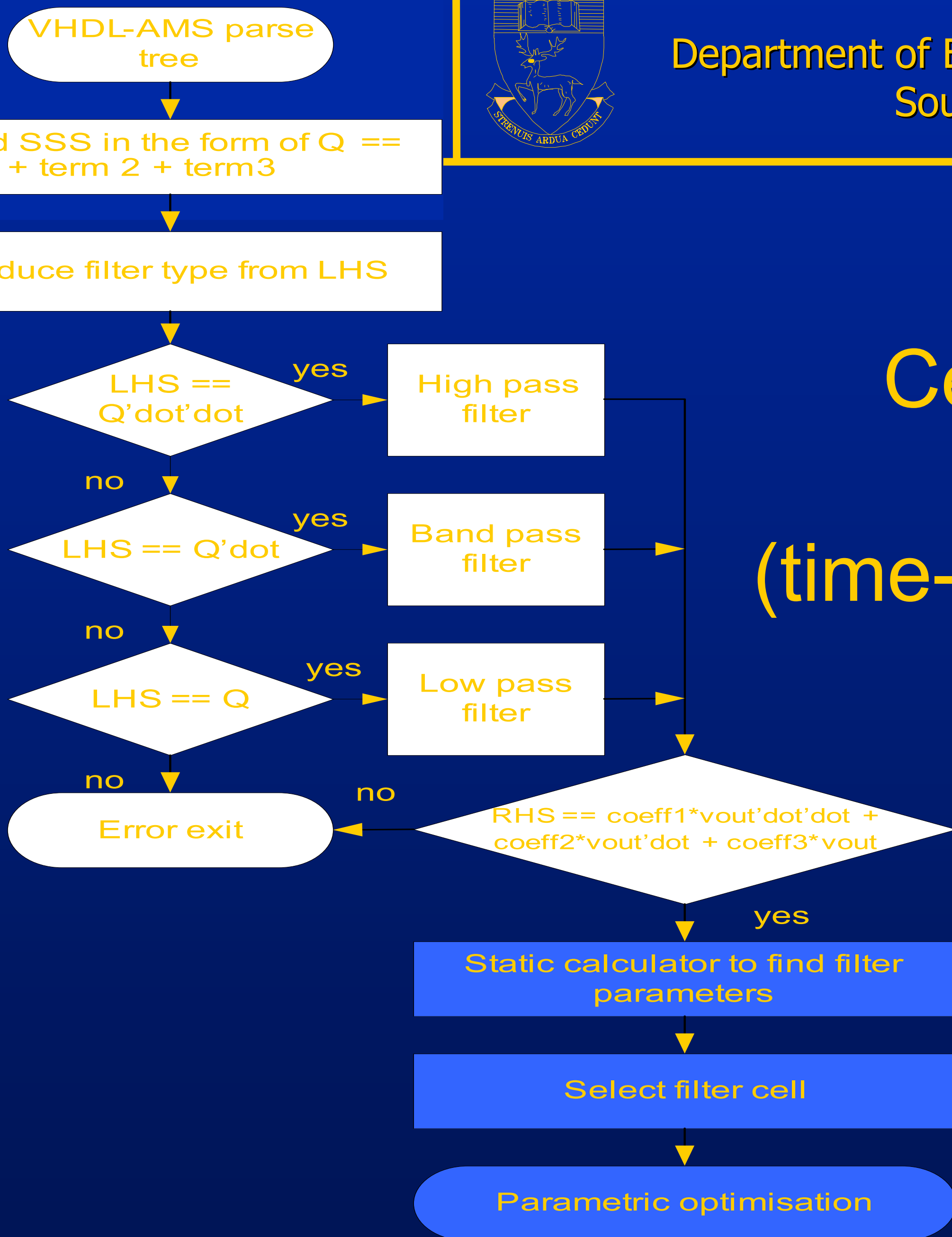
time domain

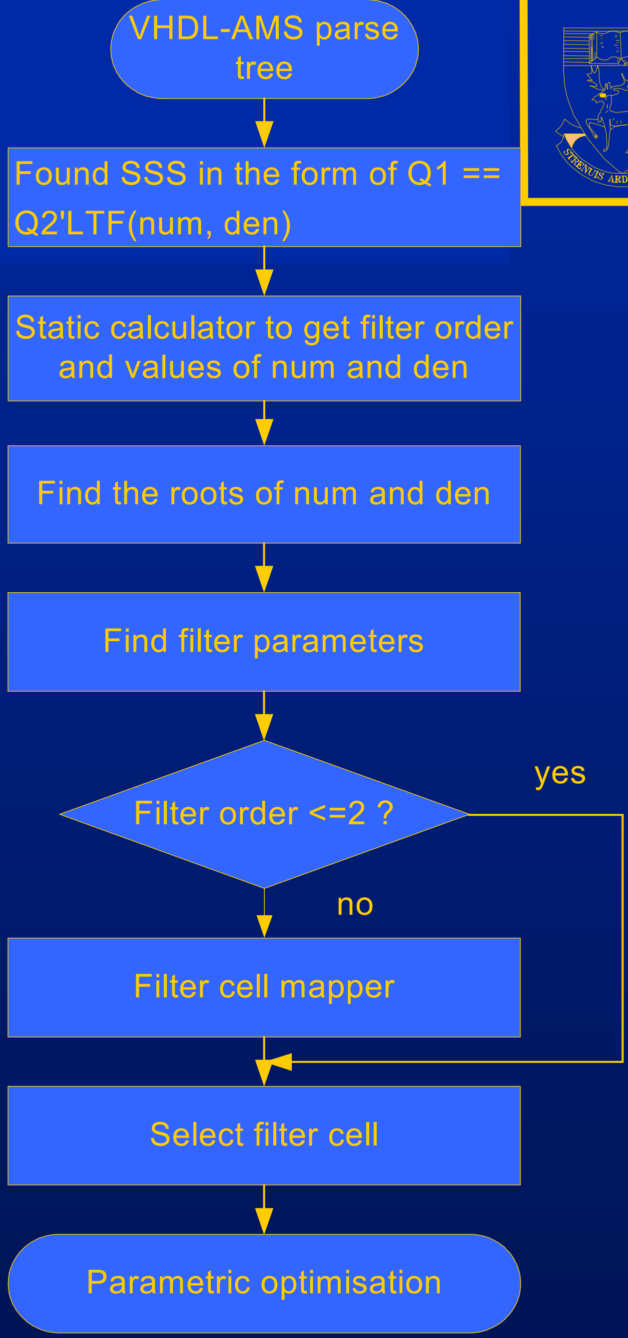
frequency domain

```
Vout == Vin'LTF(num,den);
```

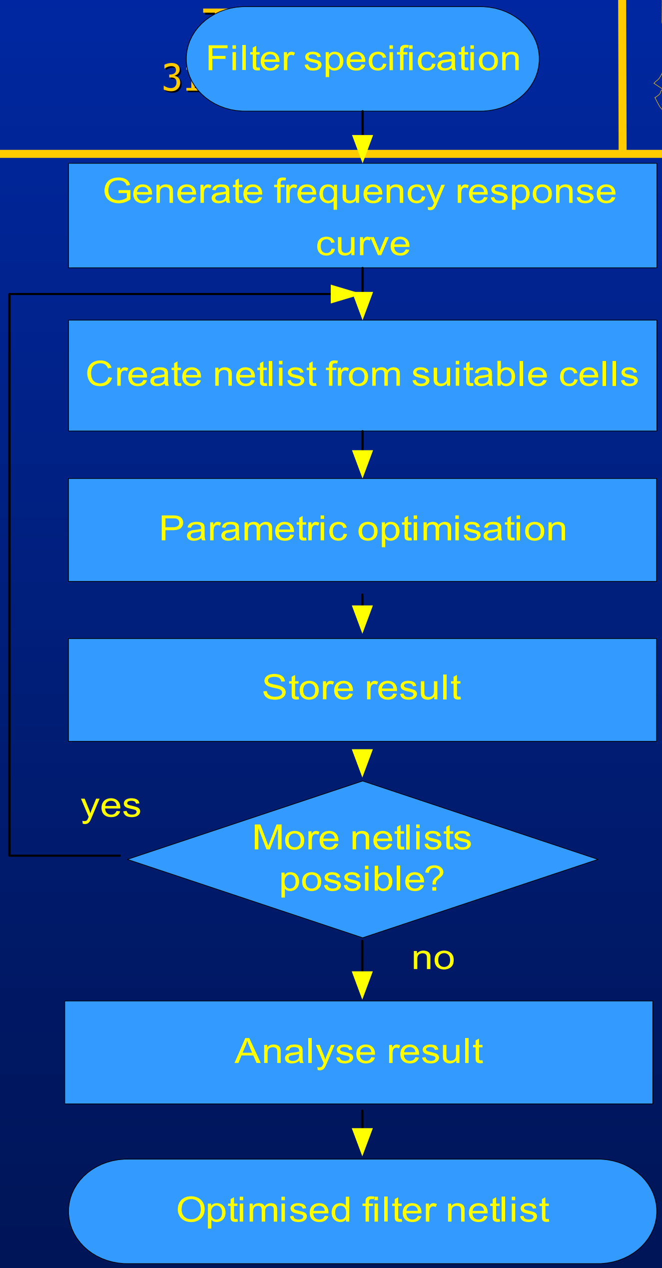
where:
constant num: real_vector:= (w*w);
constant den: real_vector:= (w*w,w/Q,1.0);

Cell selection process (time-domain model)





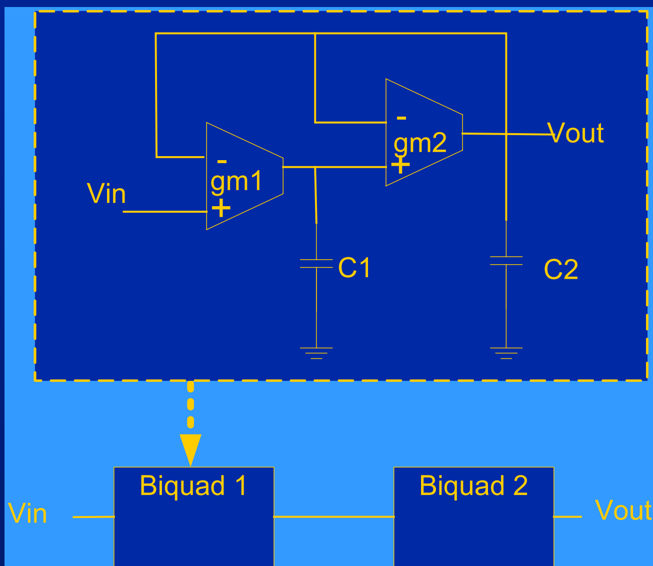
Cell selection and synthesis process for frequency domain model descriptions



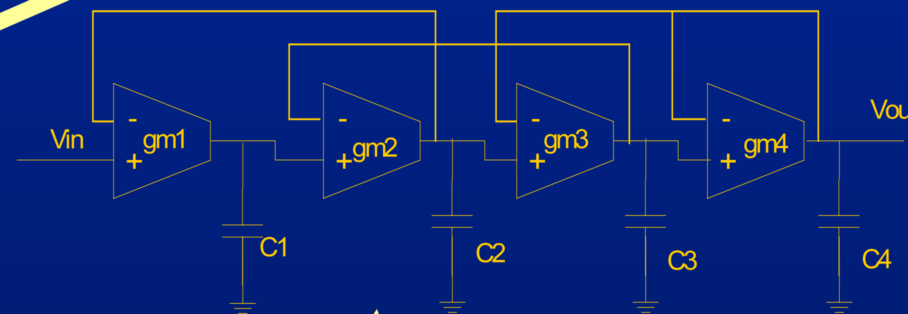
FIST optimisation strategy

Case study 1: 4th-order low-pass 1GHz

configurations selected by synthesiser

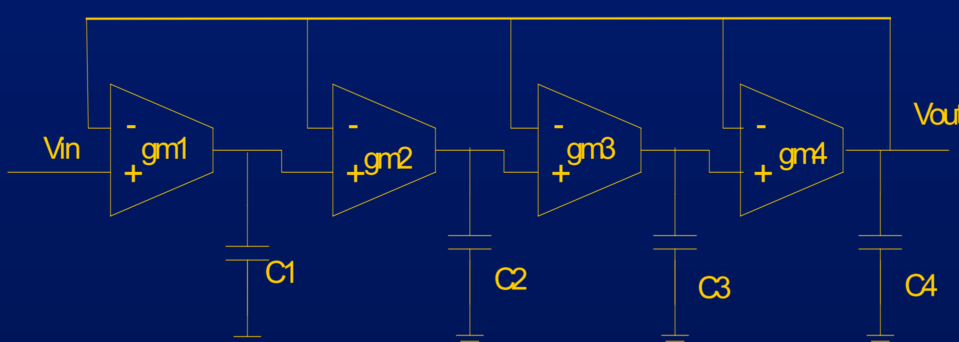


1. Simple OTA cascade
2. Wide-swing OTA cascade



3. LF with simple OTA (BEST)

4. LF with wide-swing OTA



5. IFLF with simple OTA
6. IFLF with wide-swing OTA

Case study 1: 4th-order low-pass 1GHz

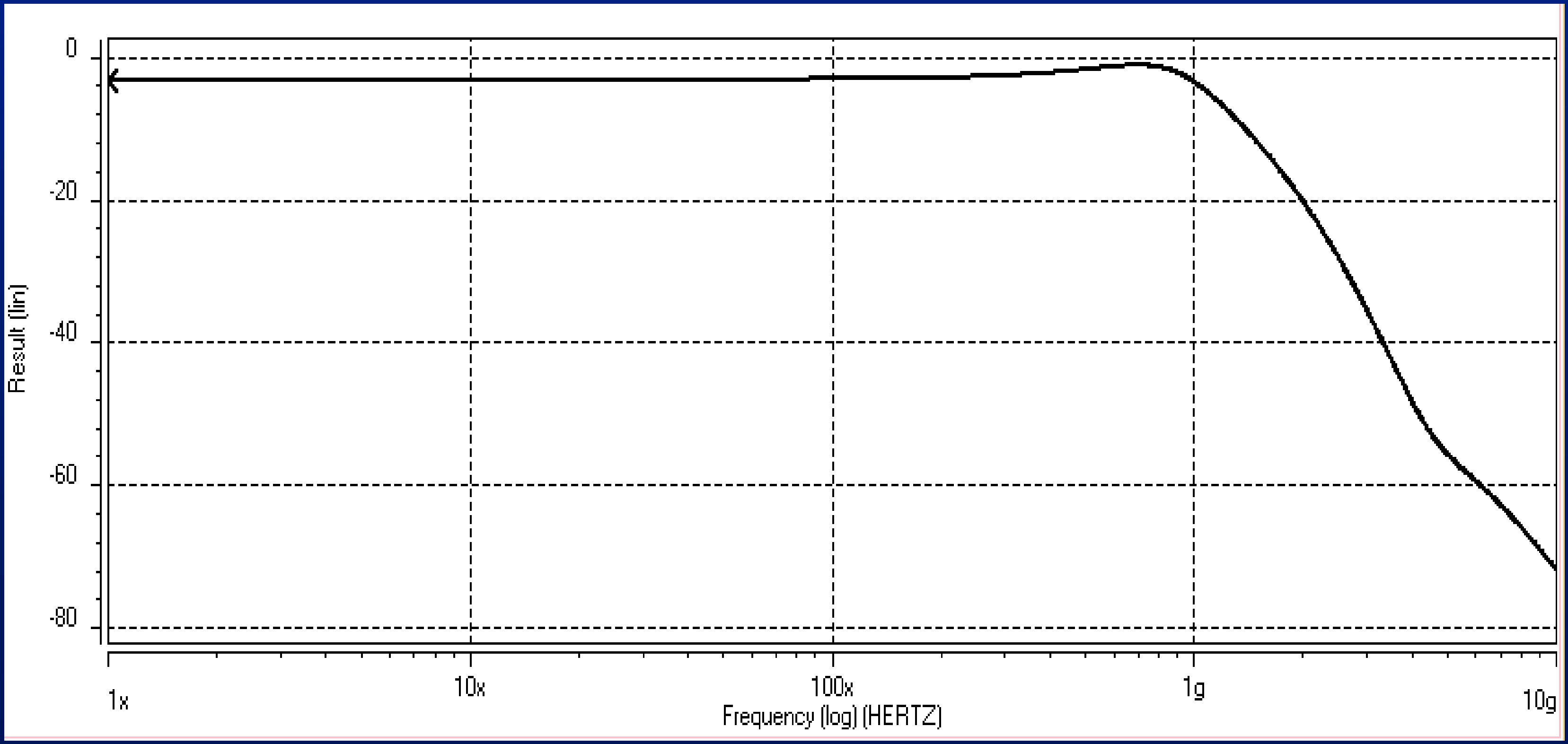
synthesis results

BEST →

	Topology	Error figure	Size (no of MOSFETs)	Power (mW)
1	Simple OTA cascade	0.663	20	160
2	Wide-swing OTA with output buffer cascade	0.513	64	2900
3	LF with simple OTA	0.307	20	163
4	LF with wide-swing OTA with output buffer	0.634	64	2900
5	IFLF with simple OTA	0.458	20	111
6	IFLF with wide-swing OTA with output buffer	380	64	420

Case study 1: 4th-order low-pass 1GHz

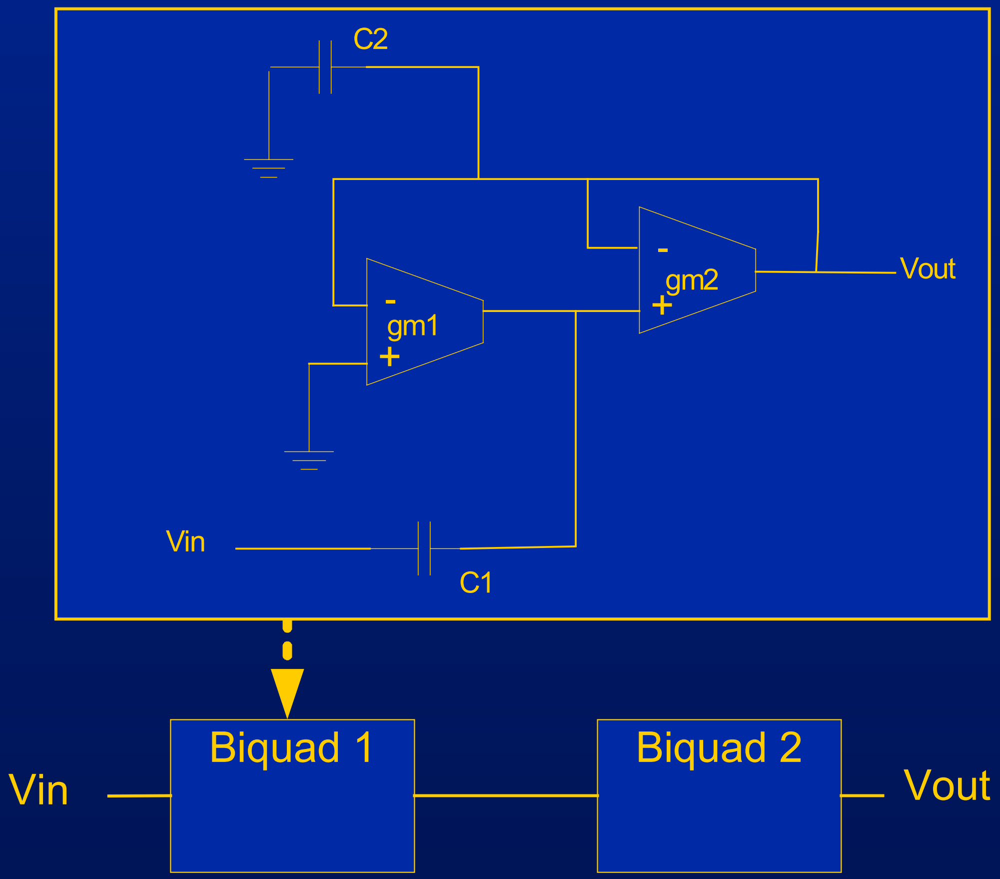
HSPICE simulation of the best candidate



Case study 2: 4th-order band-pass 1GHz

configurations selected by synthesiser (1)

1. **Wide-swing OTA cascade (BEST)**
2. Folded-cascode OTA cascade
3. Wide-swing folded-cascode OTA cascade

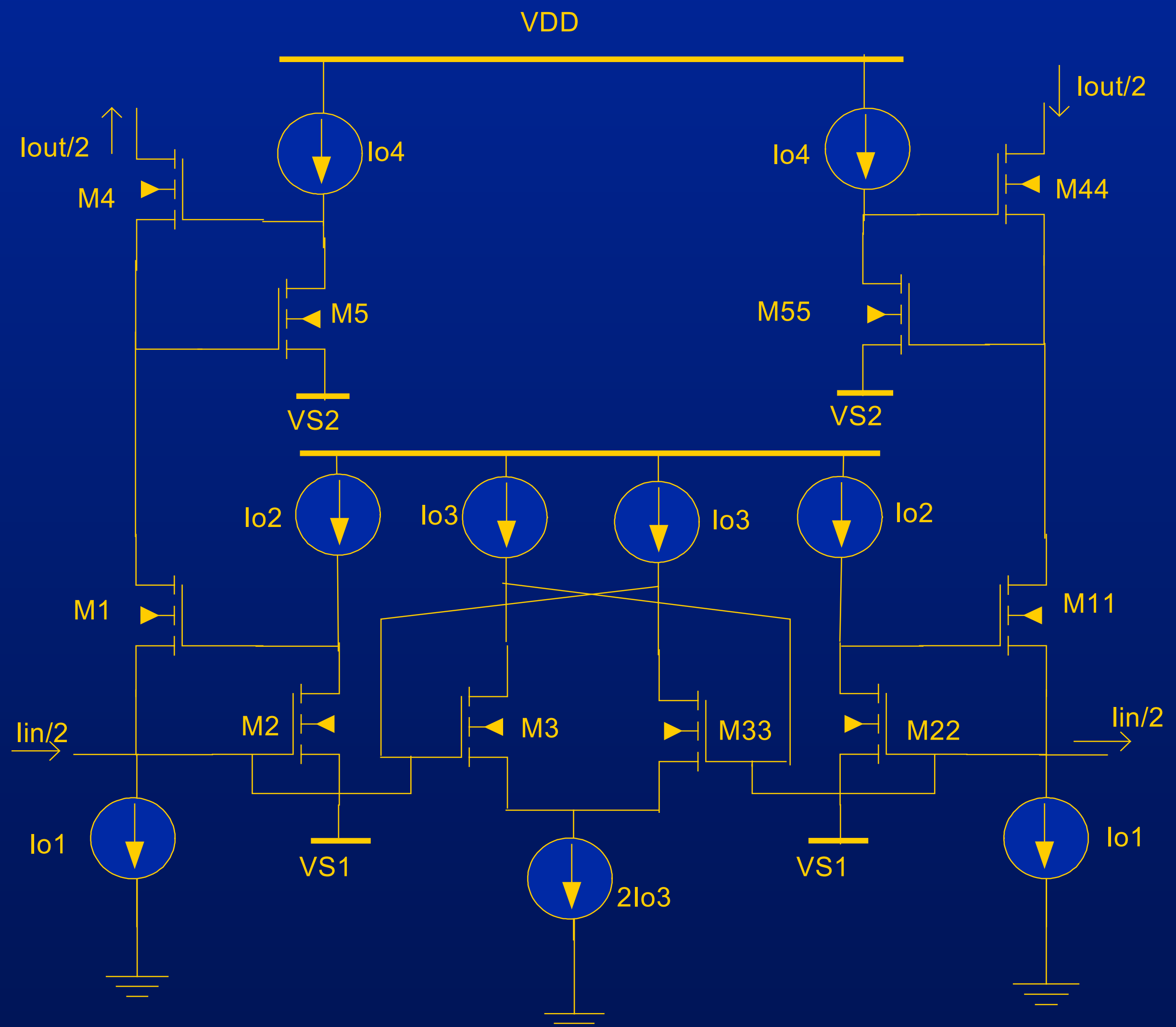


Department of Electronics and Computer Science
Southampton University

Case study 2: 4th-order band-pass 1GHz

configurations selected by synthesiser (2)

4. Vertical cascode

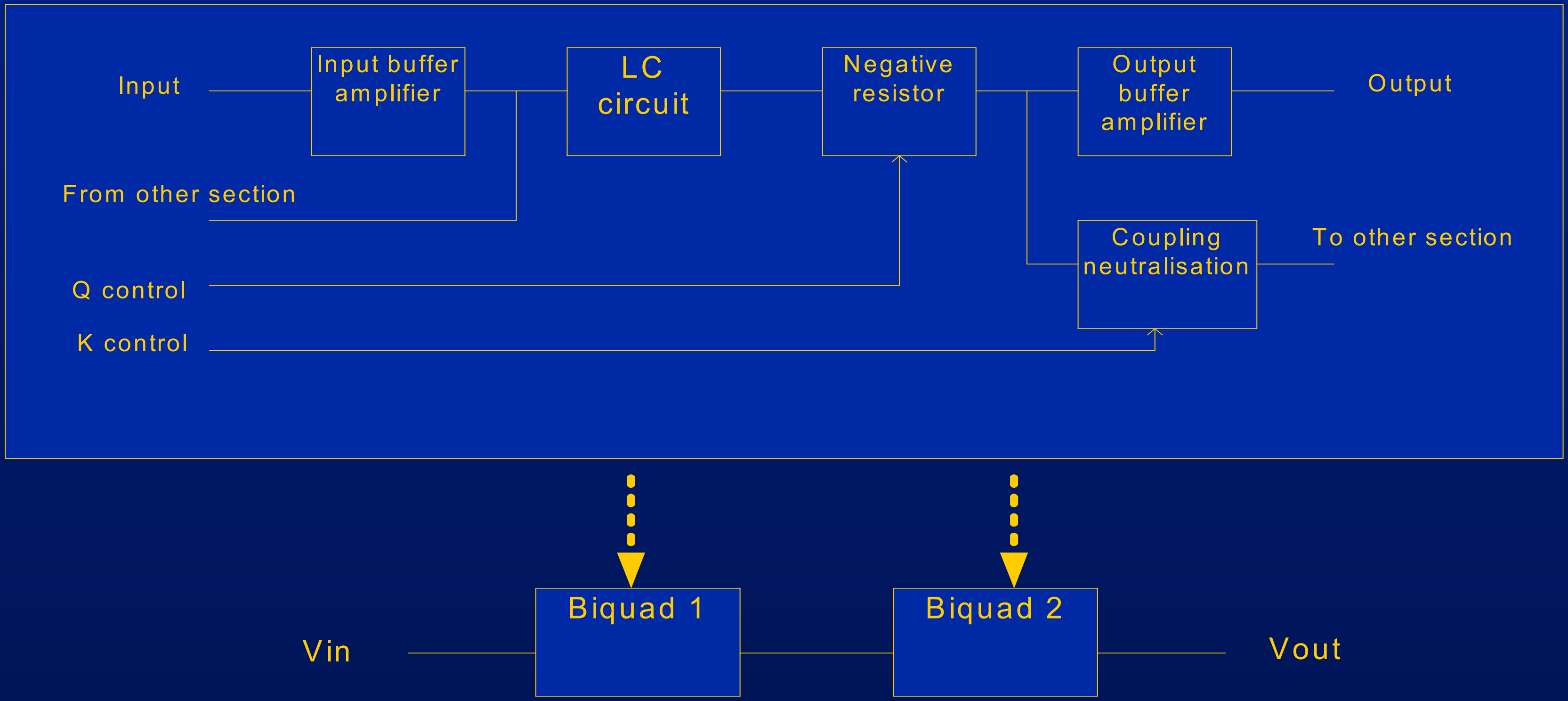


Department of Electronics and Computer Science
Southampton University

Case study 2: 4^{th}-order band-pass 1GHz

configurations selected by synthesiser (3)

5. LC (coupled resonator with a silicon spiral inductor)

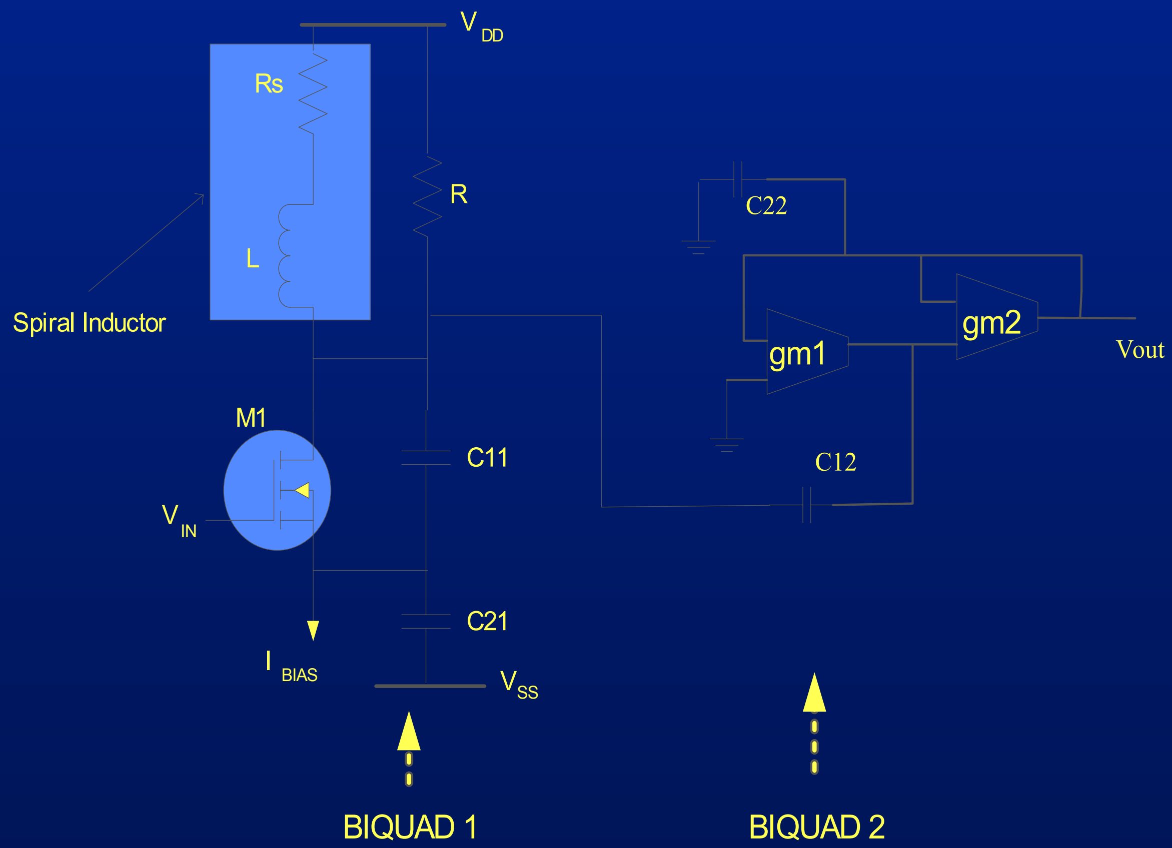


Department of Electronics and Computer Science
Southampton University

Case study 2: 4th-order band-pass 1GHz

configurations selected by synthesiser (4)

6. LC + OTA-C



Case study 2: 4th-order band-pass 1GHz

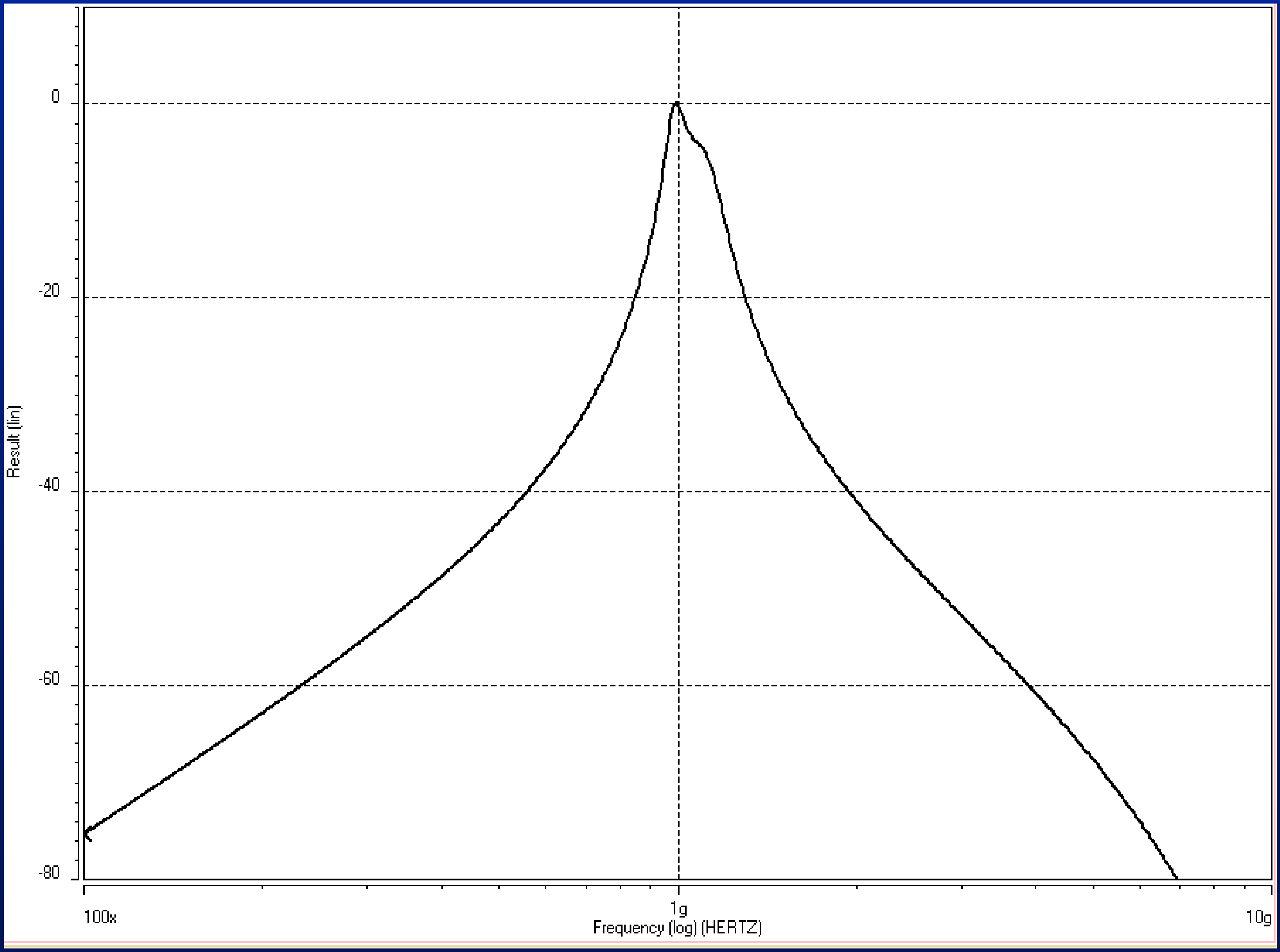
synthesis results

BEST →

	Topology	Error figure	Size (no of MOSFETs)	Power (mW)
1	Wide-swing OTA cascade	0.0611	48	137
2	Folded-cascode OTA cascade	0.0658	48	266
3	Wide-swing folded cascade OTA cascade	2.274	74	475
4	Vertical cascade	0.0688	10	484
5	LC	2.999	80	5456
6	LC-OTA-C	0.0833	25	32

Case study 2: 4th-order band-pass 1GHz

HSPICE simulation of the best candidate

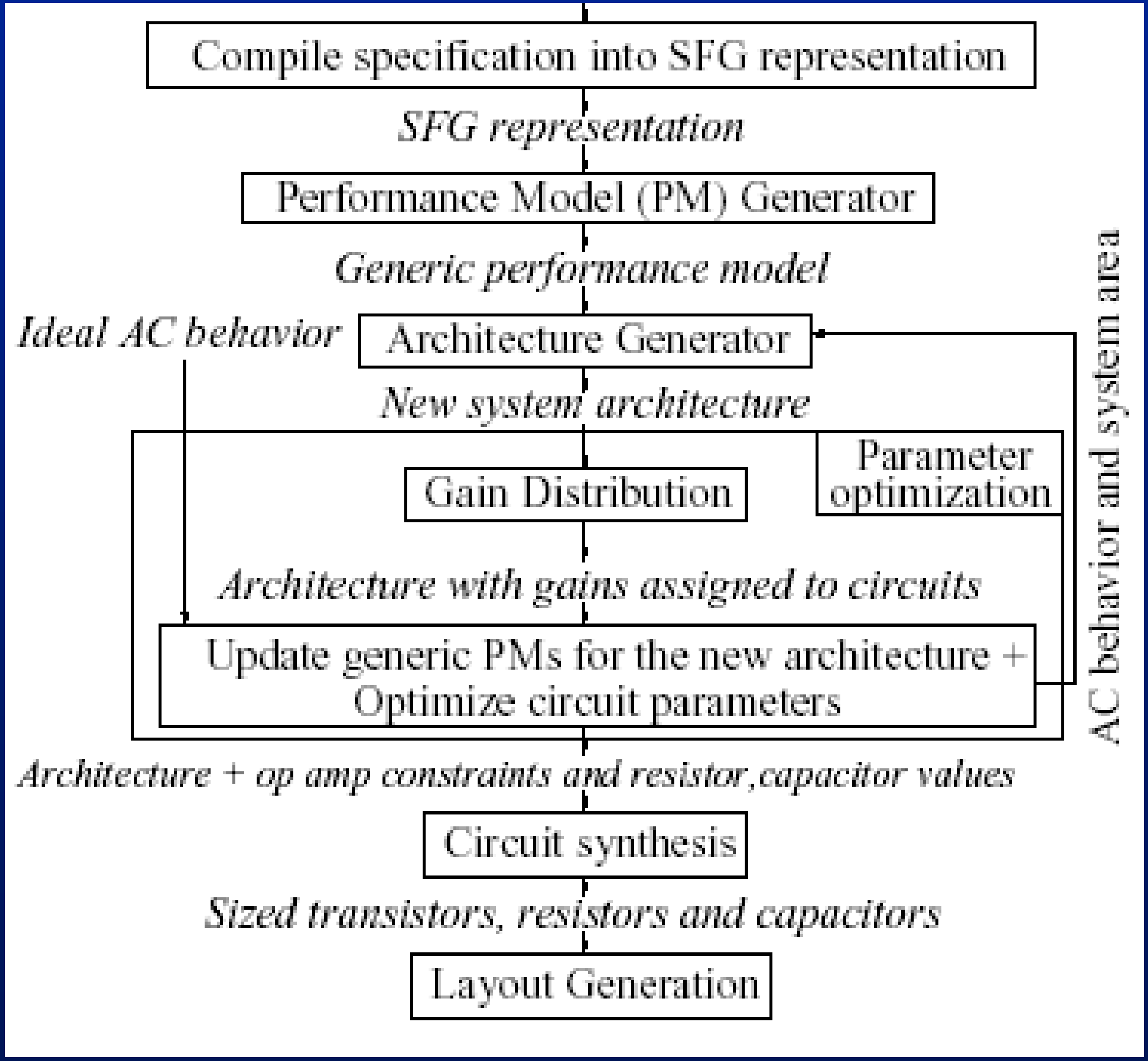


Department of Electronics and Computer Science
Southampton University

Mixed-signal high-level synthesis (HLS) from VHDL-AMS

- HLS accepts abstract specifications as inputs, explores possible architectures, and produces optimized implementations.
- Without proper HLS tools, it will be impossible to address the increasing productivity gap, and benefit from upcoming technological advancements.
- Existing digital HLS methods are obviously insufficient for mixed-signal SoC design.
- New HLS approaches are needed for *mixed-domain* specification, trade-off exploration, and integration.

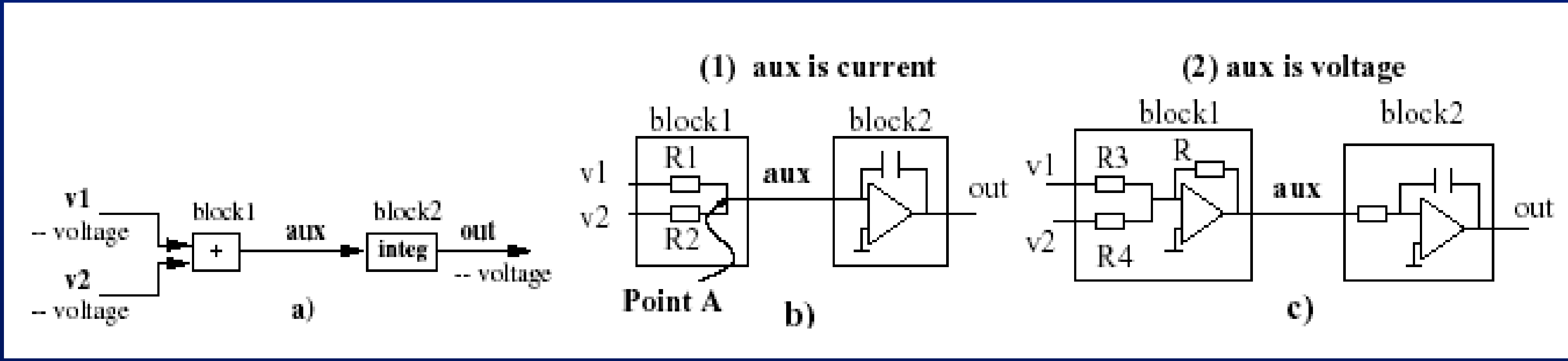
VASE high-level synthesis flow



Source: Doboli &Vemuri, *IEEE Trans CAD.* v. 22, Nov 2003

Department of Electronics and Computer Science
Southampton University

Architecture generation for signal processing in HLS based on signal flow graphs (SFG)



SFG

Structure 1

Structure 2

Department of Electronics and Computer Science
Southampton University

Present limitations of high-level MS synthesis

- Present mixed-signal high-level synthesis environments are still limited to certain types of applications, such as filters, A/D converters, or signal conditioning systems.
- HLS can tackle only applications operating at low/medium frequencies and having mild performance constraints.
- To extend mixed-signal HLS towards high-performance applications, research in several key directions is needed.

High-level AMS synthesis – future research directions

- **Modelling and specification languages for mixed-signal HLS**
  - Future specification languages should permit high-level description of various systems (PLL, oscillators, transceivers etc), express both analogue and digital functionality and constraints, and provide sufficient insight for automated generation of alternative architectures
- **Architecture generation for nonlinear analogue and RF systems**
  - New architecture generation methods are needed for creating system architectures of tightly connected blocks.
- **Performance modelling**
  - Multi-objective performance specification is required involving factors such as silicon area, speed, accuracy, low power, low voltage and hardware/software trade-offs.
- **Analogue and digital IP core integration**.
  - This should include trade-off exploration between analogue and digital domains, HLS under digital noise constraints, analogue and digital module selection, floor planning, and power net routing.

AMS synthesis - Looking further into the future

- Emergence of VHDL-AMS provides a basis for a new approach to architectural analogue synthesis.
- Topology compilers can be developed to translate high-level behaviour from VHDL-AMS to structural netlists
- First VHDL-AMS synthesis environments (FIST, NEUSYS, VASE) have been developed.
- VHDL-AMS based synthesis is likely to reduce the need for hand-crafted topologies required by many existing synthesis tools.
- Markets for VHDL-AMS tools are likely to be driven by analogue and mixed-signal synthesis applications.

Analogue and mixed-signal behavioural synthesis from VHDL-AMS

Tom J Kazmierski
Department of Electronic and Computer Science
University of Southampton, United Kingdom
tjk@ecs.soton.ac.uk, http://www.syssim.ecs.soton.ac.uk