Verilog-AMS Language

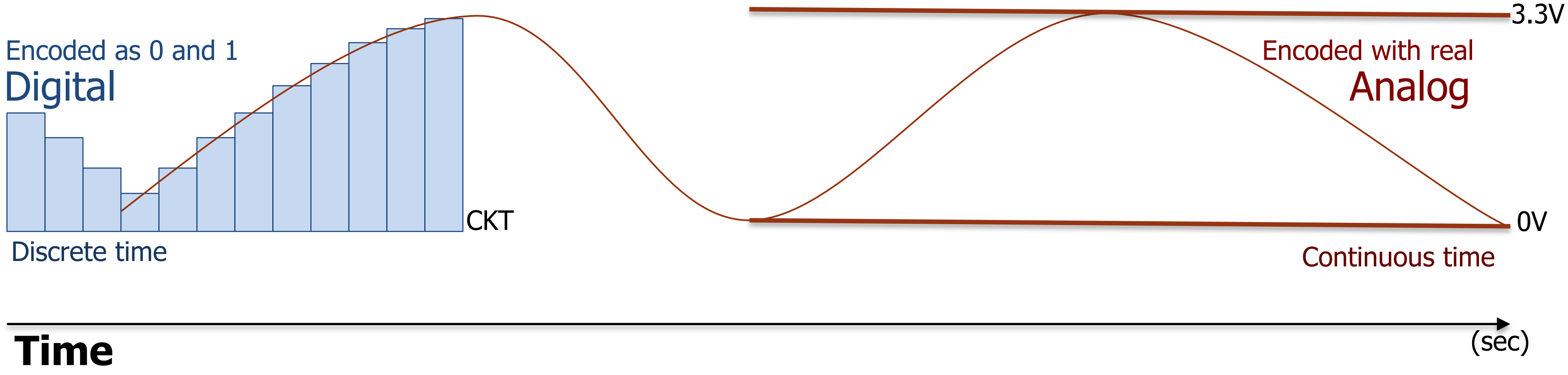
Nicola Dall'Ora
Franco Fummi

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Contents

- Introduction
- Language constructs
- Language functions
- Direct current motor example
- Assignment

Analog vs Digital (1/2)



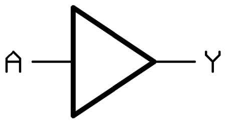
Analog vs Digital (2/2)

Logic Operator
(NOT)
Truth Table

A	**Y** (!A)
0	1
1	0

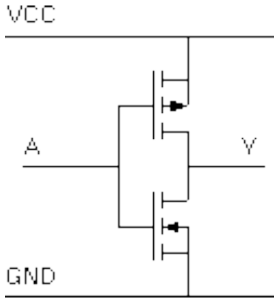
Represents the negation of a value as Boolean function

Digital
Circuit

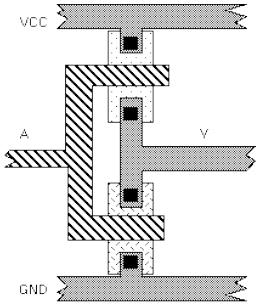


Graphical representation,
Implements a boolean function

Analog
Circuit



Polygon
Layout



Analog Behavior
V(Y) = VCC – V(A)

Abstraction Level

Hardware Description Languages (HDL)

Digital world

- Verilog
- VHDL
- System-Verilog
- SystemC

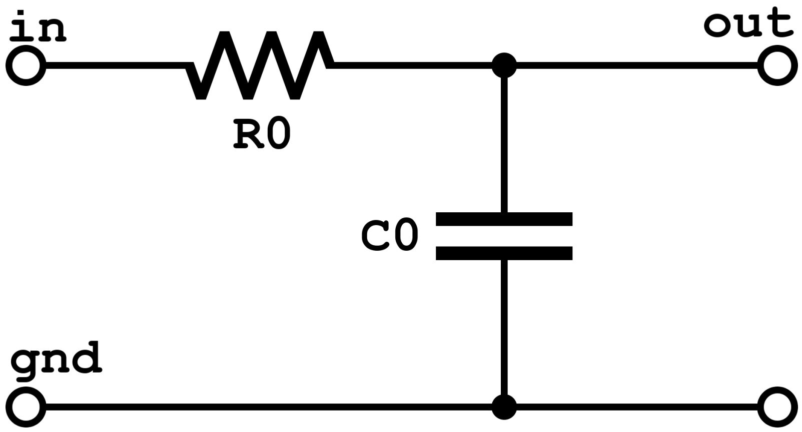
Analog world

- Verilog-AMS
- VHDL-AMS
- System-Verilog AMS (standard under definition)
- SystemC-AMS
- Spice
- Spectre / Eldo

Circuit Structure

```
`include "disciplines.vams"

module rc_block(in, out, gnd);
  input in, gnd;
  output out;
  electrical in, out, gnd;
  parameter real R0 = 100.0;
  parameter real C0 = 0.0001;
  analog begin
    I(in, out) <+ V(in, out) / R0;
    I(out, gnd) <+ ddt(V(out, gnd)) * C0;
  end
endmodule
```

equations of the model



Code Structure

```
`include "disciplines.vams"

module rc_block(in, out, gnd);
  input in, gnd;
  output out;
  electrical in, out, gnd;
  parameter real R0 = 100.0;
  parameter real C0 = 0.0001;
  analog begin
    I(in, out) <+ V(in, out) / R0;
    I(out, gnd) <+ ddt(V(out, gnd)) * C0;
  end
endmodule
```

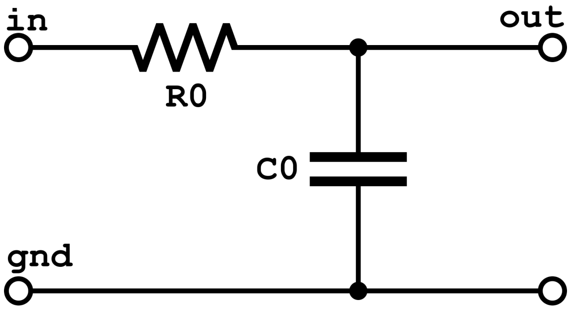
Interface

Ports

Nodes Type

Parameters

Behavior



Branch Contributions

The line:

```
I(in, out) <+ V(in, out) / R0;
```

defines the relationship between the module ports **out** and **in**. This is known as a **branch contribution** and is one of the most important Verilog-A concepts.

Both **V() and I() functions** in the above are known as an **access function**.

- V(): provides the **potential difference between the two nodes**
- I(): provides the **current flowing between its two nodes**

Letter V stands for Voltage, while I is the Current (letter I comes from the French, *intensité de courant*)

Disciplines (1/2)

The line:

```
electrical in, out, gnd;
```

defines the **discipline** for the module ports and ground node in this case '**electrical**'.

Verilog-AMS supports other disciplines such as **thermal**, **mechanical** and **rotational** allowing **simulation** of **physical processes** other than electrical and electronic.

The definitions of these other disciplines are defined in the disciplines.vams file which is included using the line:

```
`include "disciplines.vams"
```

Disciplines (2/2)

A list of disciplines supported by Verilog-AMS:

Name	Potential	Flow	Domain
logic	–	–	discrete
electrical	Voltage	Current	continuous
thermal	Temperature	Power	continuous
kinematic	Position	Force	continuous
rotational	Angle	Angular_Force	continuous

With 2 variables (Voltage and Current for electrical) we can simulate our circuit

Natures

Potential and **Flow** of disciplines are selected from a table of **Natures**

Name	Units	Access
Voltage	V	V
Current	A	I
Charge	coul	Q
Temperature	K	Temp
Position	m	Pos
Velocity	m/s	Vel
Acceleration	m/s^2	Acc
Impulse	m/s^3	Imp
Force	N	F
Angle	rads	Theta
Angular_Force	N-m	Tau

Functions

The line:

time derivative

```
I(out, gnd) <+ ddt(V(out, gnd)) * C0; // Simple capacitor beh. model
```

contains a function called **ddt**. An **analog operator** that performs the time derivative of the passed argument. There are many other analog operators in Verilog-AMS.

Operator	Description
`ddt(operand, [abstol|nature])`	Time derivative
`idt(operand, [ic], [assert], [abstol|nature])`	Time integral
`transition(operand, delay, trise, [tfall])`	Transition
`slew(operand, [rising_sr], [falling_sr])`	Slew
`absdelay(operand, delay, [max_delay])`	Delay
`laplace_zp(operand, [zeta], [rho], [epsilon])`	Laplace, zero-pole form
`laplace_nd(operand, [n], [d], [epsilon])`	Laplace, numerator-denominator form
`last_crossing(operand, [direction])`	Last crossing
`limexp(operand)`	Limited exponential

Timing Statements – Analog Event (1/2)

- Force the analog kernel to place an evaluation point at a particular point in time
- Optionally execute a statement at that instant of time.

The code:

```
analog begin
     @(timer(0, T)) hold = V(in);
     V(out) <+ transition(hold, 0, 100n);
end
```

In this example, the simulator will stop at t=0, t=T, t=2T, etc. and at each of those points it will execute ‘hold = V(in)’. It will not execute this statement at any other points in time.

Timing Statements – Analog Event (2/2)

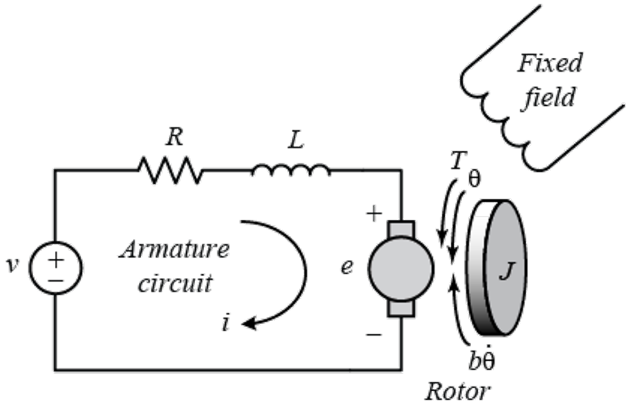
Function	Description
timer(*time*[, *period*][, *ttol*][, *enable*])	The event function generates events at particular instants of time and forces the simulator to evaluate the circuit at those points.
cross(*wave*[, *dir*][, *ttol*][, *tol*][, *enable*])	The first argument to *cross* is an expression whose value varies with time. The *cross* function generates events when the value of the expression crosses 0.
above(*wave*[, *ttol*][, *tol*][, *enable*])	*above* is similar to *cross* except it operates in both DC and IC analyses

Direct Current (DC) Motor Model

- DC motor modelling as an **equivalent circuit** of the armature and the **free-body diagram** of the rotor

$$v = K_M \cdot \omega + R \cdot i + L \cdot \frac{di}{dt}$$

$$\tau = K_T \cdot i - d \cdot \omega - j \cdot \frac{d\omega}{dt}$$



- **Input** of the system is the voltage source (V) applied to the motor's armature
- **Output** of the system the rotational speed of the shaft $\dot{\theta}$
- The rotor and shaft are assumed to be **rigid**
- The **friction torque** is proportional to shaft angular velocity

Equations of the DC Motor

```
Discipline rotational_omega
  potential Angular_Velocity;
  flow Angular_Force;
enddiscipline
```

```
// Angular Velocity in radians per second
Nature Angular_Velocity
  units = rads/s
  access = Omega
```

```
// Force in newtons
Nature Angular_Force
  units = n/m
  access = Tau
```

```
Discipline electrical
  potential Voltage;
  flow Current;
enddiscipline
```

Motor winding inductance (H)

Current in the branch p-n

Motor winding resistance (Ohms)

Time derivative of current in the branch p-n

Angular velocity of the shaft

Motor constant (V·s/rad)

$$V(p,n) \mathrel{<+} \underbrace{km * Omega(shaft)}_{V} + \underbrace{r*I(p,n)}_{R1} + \underbrace{l*ddt(I(p,n))}_{L1}$$

Inertia of shaft (N·m·s/rad)

Angular Velocity of the shaft

Time derivative of the angular velocity of the shaft

Drag (friction) N·m·s/rad

Current in the branch p-n

Flux constant (N·m/A))

$$Tau(shaft) \mathrel{<+} kf*I(p,n) - d*Omega(shaft) - j*ddt(Omega(shaft))$$

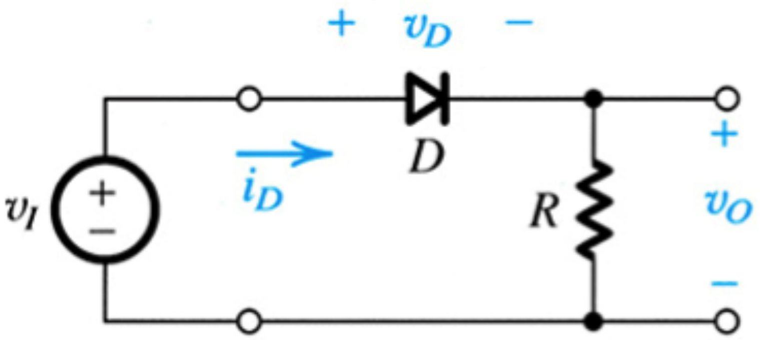
Verilog-AMS Code - DC Motor

```
`include "disciplines.vams"
`include "constants.vams"
`timescale 1us / 1us

module motor(shaft_position, p, n);
    // PARAMETERS -----------------------
    parameter real km = 4.5;  // motor constant (V-s/rad)
    parameter real kf = 6.2;  // flux constant (N-m/A)
    parameter real j = 1.2;   // inertia of shaft (N-m-s2/rad) 0.004
    parameter real d = 0.1;   // drag (friction) (N-m-s/rad)
    parameter real r = 5.0;   // motor winding resistance (Ohms)
    parameter real l = 0.02;  // motor winding inductance (H)
    // PORTS ---------------------------
    output shaft_position;//, i_absorb;
    input p, n;
    // NODES ---------------------------
    rotational shaft_position, rognd;
    // electrical i_absorb;
    electrical p, n;
    // Internal nodes.
    electrical n1, n2;
    rotational_omega shaft, rgnd;
    // Reference nodes.
    ground rgnd, rognd;
    // BRANCHES ------------------------
    branch (p, n1) Vm;
    branch (n1, n2) R1;
    branch (n2, n) L1;
    branch (shaft, rgnd) bshaft;
    branch (shaft_position, rognd) bshaftp;

    // BEHAVIOR ------------------------
    analog begin
        // Electrical model of the motor winding.
        V(Vm) <+ km * Omega(bshaft);
        V(R1) <+ r * I(R1);
        V(L1) <+ l * ddt(I(L1));
        // Physical model of the shaft (keep like this).
        Tau(bshaft) <+ + kf * I(Vm);
        Tau(bshaft) <+ - d * Omega(bshaft) - j * ddt(Omega(bshaft));
        // Equation for conversion to degrees.
        Theta(bshaftp) <+ (180 * idt(Omega(bshaft), 0)) / `M_PI;
        // deg : rad = 180 : 3.14
        // deg = 180 * rad / 3.14
    end
endmodule
```

How to Compile and Simulate

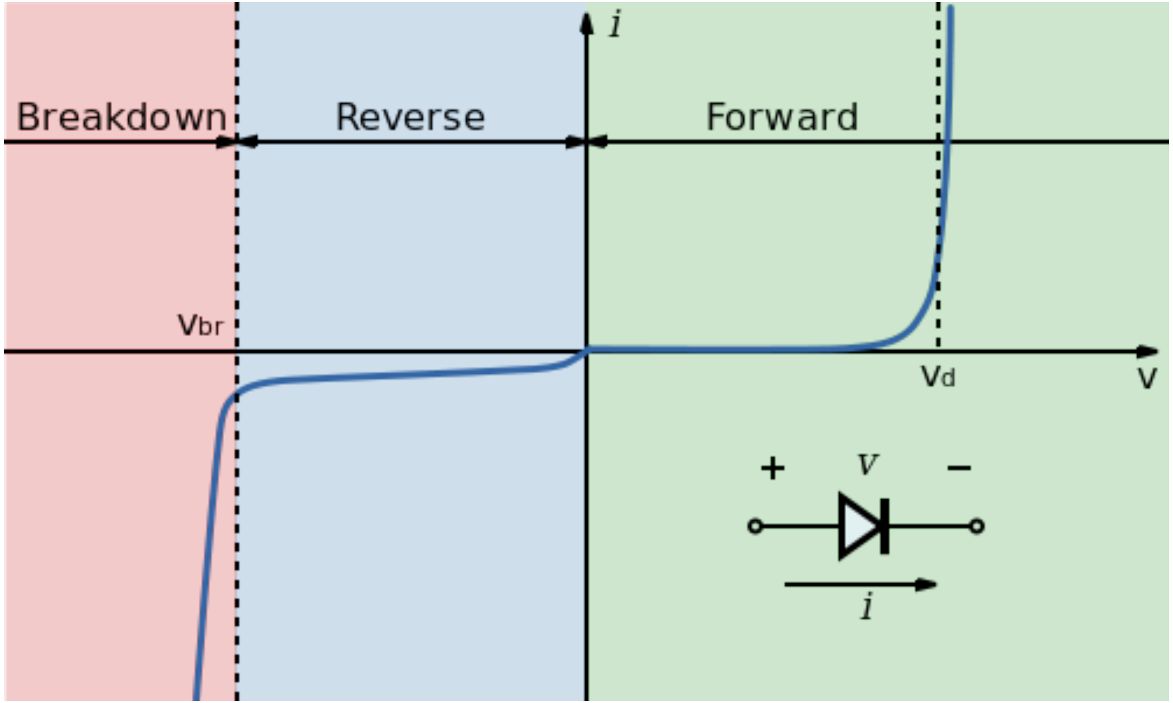
- Procedure tested on Ubuntu 18.04 LTS
- Download the source code from the e-learning (07_sources.zip)
- Download and install Docker (https://phoenixnap.com/kb/how-to-install-docker-on-ubuntu-18-04)
- Download and install PulseSecure to connect to the university VPN
- Download the simulator compiled files (https://www.dropbox.com/s/g9iv0io4mi7wj7l/mentor.tar.bz2?dl=0), decompress it (`tar –xvf archive.tar.bz2`) and save it in the /opt/mentor-centos system folder
- Modify the path to the simulator compiled files in the DCMotor/docker/run.sh file
- Run centos7 on docker, from the source code DCMotor/docker, first launch build.sh and then run.sh
- From the docker shell, move to the source code of the DCMotor
- To compile the Verilog-AMS code: `make compile`
- To simulate the Verilog-AMS code with QuestaSim: `make batch` or `make sim`
- To check the simulation result: `ezwave run-eldo/module.wdb`

Assignment: Simple Diode Circuit



- Model this simple electrical circuit in Verilog-AMS, and try to simulate it for 10 *second* with a voltage source of 3 *Volts.*
- Find a correct parameter value for the diode and for the resistor to obtain the correct behavior of the diode in output.
- Analyze the simulation behavior with ezwave
- Describe all the steps in a short report (1/2 pages)

Behavior of a Diode



References

- https://verilogams.com/refman/index.html
- https://designers-guide.org/

Questions?

nicola.dallora@univr.it