Verilog-A Overview

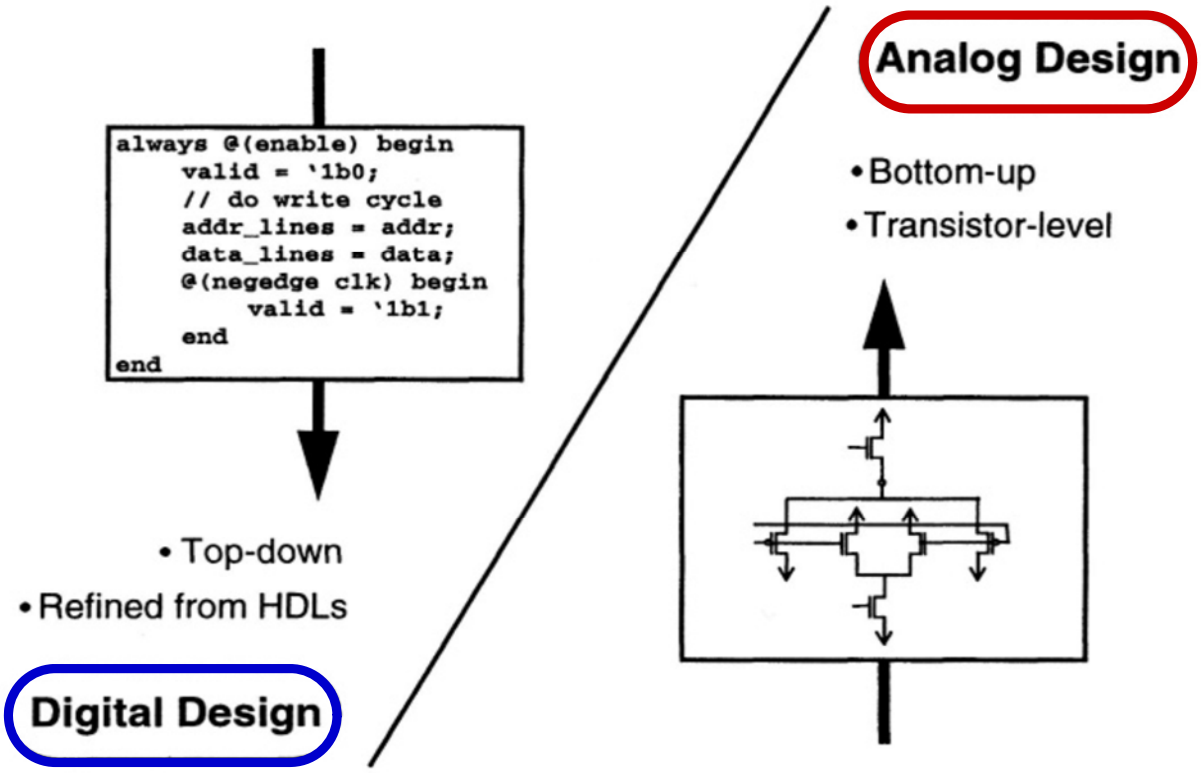
Prof. Chien-Nan Liu
Institute of Electronics
National Chiao-Tung Univ.

Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
http://mseda.ee.nctu.edu.tw/jimmyliu

Outline

- Introduction
- Analog system description
- Data types and declarations
- Behavioral descriptions in Verilog-A
- Math functions and environment parameters
- References

Difference between Digital and Analog Design



Modeling Languages

- Programming languages:
  - FORTRAN (SPICE2)
  - C (SPICE3)
    - Fast, direct access to simulator
    - Must compute derivatives
    - No standard interface
      - Intimate knowledge of simulator required
- MATLAB
  - Excellent for data fitting
  - Does not run directly in any analog simulator

Behavioral Modeling Languages

- VHDL-AMS
  - First analog behavioral modeling language working group (IEEE 1076.1)
  - Painfully slow to come to fruition …
  - Europe prefers VHDL (for digital)
  - Runs in:
    - AMS Designer (Cadence), DiscoveryAMS (Synopsys), ADVance MS (Mentor), Smash (Dolphin), …
    - only AMS simulators!
  - No clear definition of "VHDL-A" (except by R. Shi's MCAST model compiler)

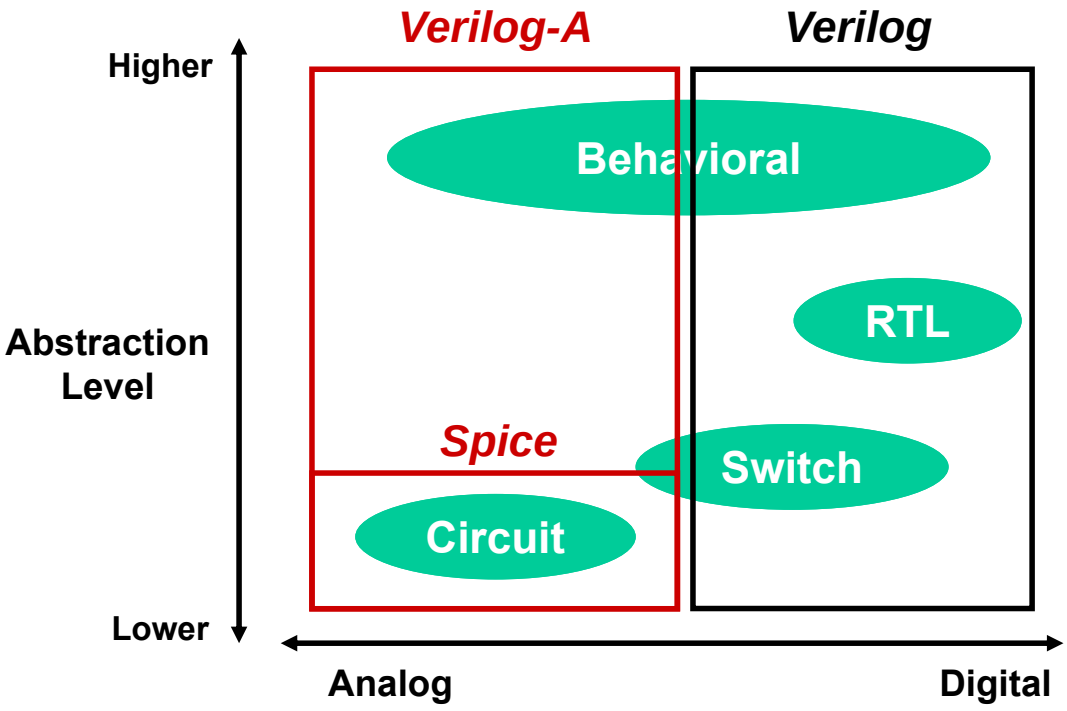
Behavioral Modeling Languages

- Verilog-A→ Verilog-AMS
  - Pushed by Cadence, came to market earlier
  - Verilog-A from Open Verilog International became part of Accellera Verilog-AMS
  - IEEE 1800 authorized to develop SystemVerilog-AMS
  - Verilog-AMS runs in the same AMS simulators as VHDL-AMS
    - Verilog-A runs in Spectre, HSpice, ADS, Eldo… and internal simulators of semiconductor companies
  - \+ Clear definition of "A"

Why Verilog-A?

- Faster implementation compared to C (or FORTRAN)
  - BSIM3 self-heating: 1-2 days in Verilog-A versus 2-3 weeks in C
  - Derivatives coded automatically
- Multiple simulator support
  - AnalogDevices: Adice,
  - Motorola/Freescale: Mica
  - Cadence: Spectre
  - MentorGraphics: Eldo
  - Synopsys: NanoSim
  - Agilent: ADS & ICCap
  - Silvaco: SmartSpice & UTMOST, …
- No* simulator-specific details

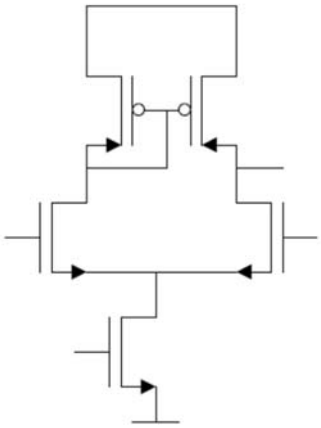
Verilog-A as an Extension of Spice



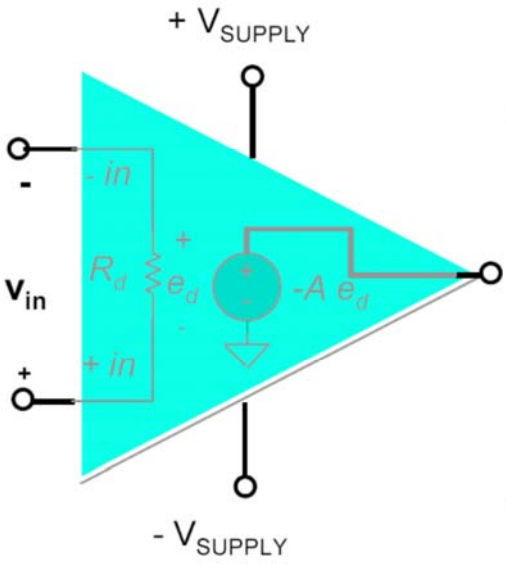
Analog Model Abstraction

- Trade-off: **complexity** and **accuracy**

Circuit level



Macro Model



Analog Behavioral

```
module opamp (vout, vin_p, vin_n );
  inout  vin_p , vin_n;
  output vout;
  electrical vin_p, vin_n, vout;


  analog begin

    I(vin_p)  <+ 0.0;
    I(vin_n)  <+ 0.0;
    V(vout) <+ V(vin_p, vin_n ) *Av;
  end
endmodule
```


Outline

- Introduction
- Analog system description
- Data types and declarations
- Behavioral descriptions in Verilog-A
- Math functions and environment parameters
- References

Basic Module Definition

Include natures, discipline & constants →

```
`include "constants.h"
`include "discipline.h"
```

Interface Declarations
name, ports and parameters →

```
module res1(p, n) ;
   inout p, n ;
   electrical p, n;
   parameter real r=1 from ( 0:inf) ;
   parameter real tc=1.5m from [0:3m) ;
```

Global Module Scope
local variables and analog block →

```
   real reff;
   analog begin
      @(initial_step("static")) begin
           reff = r*(1+tc*$temperature) ;
      end
   I(p,n) <+ V(p, n)/reff ;
   end
endmodule
```


Predefined Conservative Disciplines

- Defined in **disciplines.h**

Disciplines	Potential			Flow		
	Nature	Access	Units	Nature	Access	Units
Electrical	Voltage	V	V	Current	I	A
magnetic	**Magnetomotive force**	MMF	A-turn	Flux	Phi	Wb
thermal	**Temperature**	Temp	°C	Power	Pwr	W
kinematics position	Position	Pos	m	Force	F	n
velocity	Velocity	Vel	m/s	Force	F	n
rotational phase	Angle	Theta	rads	Torque	Tau	n/m
velocity	**Angle Velocity**	Omega	rads/s	Torque	Tau	n/m

Basic Analog Model : nature

- Signal type declarations
  - Used in discipline declarations or other nature declarations
  - Specifies a set of attributes associated with a signal type
- Required attributes
  - Absolute tolerance(real number)
  - units(string)
  - access function(name)
- Optional user and simulator defined attributes

Voltage

```
nature Voltage
  abstol = 1u ;
  units  = "v" ;
  access = V;
endnature
```

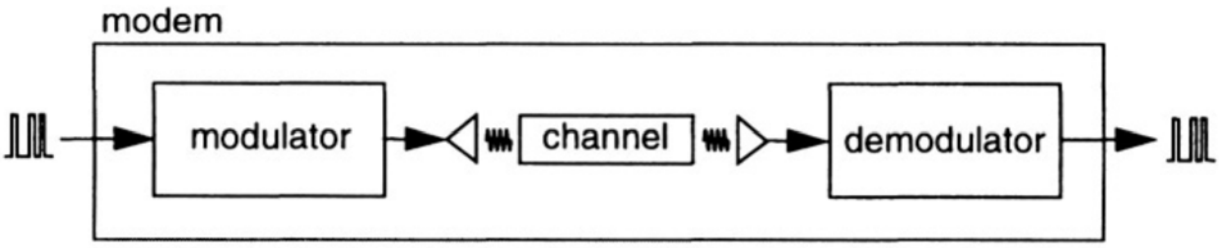
Current

```
nature Current
  abstol = 1p ;
  units  = "A" ;
  access = I;
endnature
```


Analog System Description and Simulation

- Structural Description
  - A module is comprised of other sub-modules
- Behavioral Description
  - Descriptions in a programmatic fashion with the Verilog-A language
  - The module is defined in terms of the values for each signal
- Mixed-level Descriptions
  - Combine both Structural and Behavioral Descriptions

Structural Description Example: Modem



The modem system is consist of :
1. modulator
2. channel
3. demodulator

Structural Description of the Modem System

```
// Verilog-A definition of the modem System

module modem(dout, din);
  inout  dout, din;          <-- Port signal declarations
  electrical dout, din;      <-- and connections
  parameter real fc = 100.0e6;   <-- Parameter declarations

  electrical clk, cin, cout;     <-- The type of analog signals

    qam_mod  #(.carrier_freq(fc))  mod(cin, din, clk);
    channel  c1( cout, cin);
    qam_demod  #(.carrier_freq(fc))  demod(dout, cout, clk);
endmodule
```

Structural Description

type of module instance

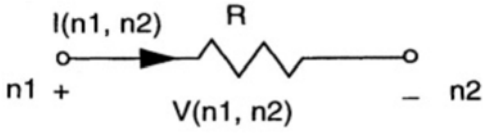
name of the instance created

```
qam_mod  #( .carrier_freq(fc))  mod(cin, din, clk);
```

parameter name in child (**qam_mod**) module assigned as: **carrier_freq = fc**

Behavioral Description

- Be encapsulated within **analog** statement
- Mathematical mappings from input signals to output signals
- *Contribution operator* "**<+**" in Verilog-A language
  - Assign an expression to a **signal**



Robust description:

V(n1, n2) <+ I(n1, n2)*R;

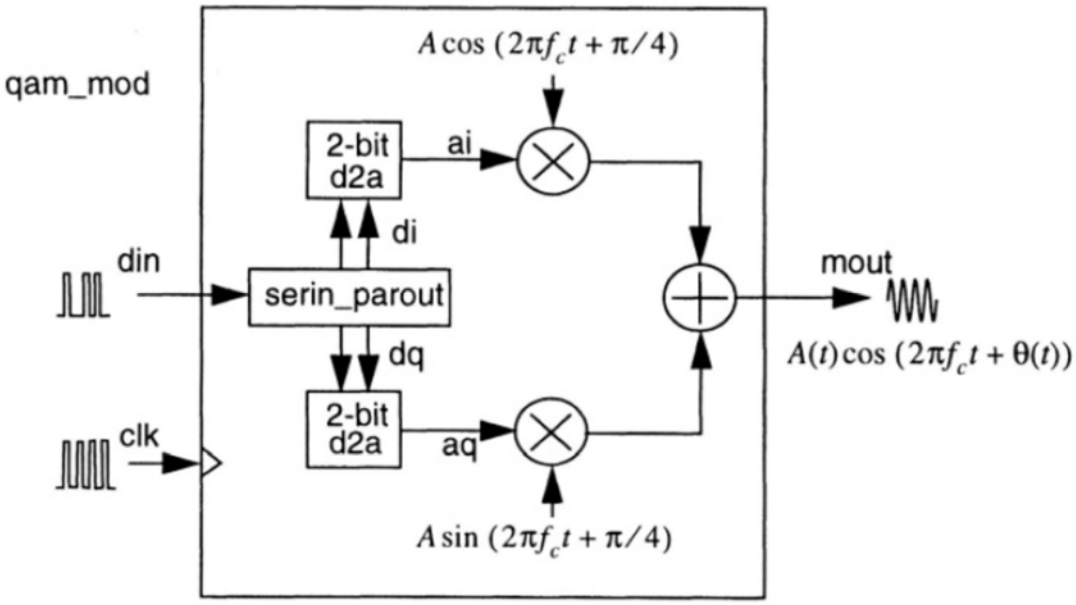
even if R is zero in some modeling cases
e.g. voltage-controlled resister

```
module resistor(n1, n2);
  inout n1, n2;
  electrical n1, n2;

  parameter real R = 1.0;

  analog

    I(n1, n2) <+ V(n1, n2)/R;

endmodule
```

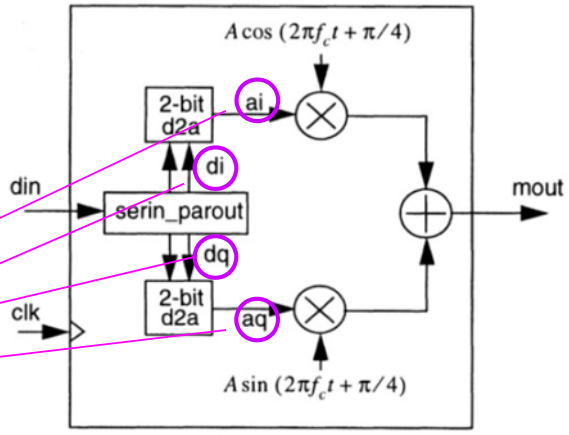
Mixed-level Descriptions

- Combine both Structural and Behavioral Descriptions

For example: 16_QAM modulator for a modem system



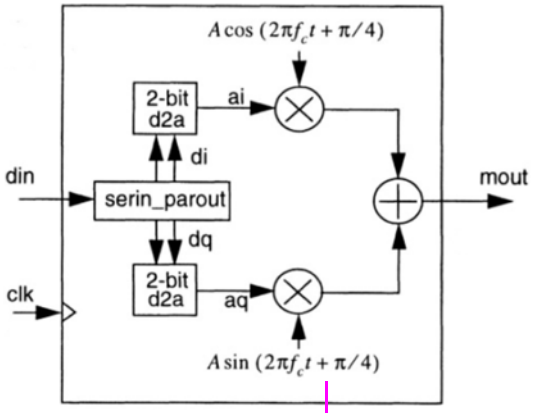
Verilog-A Mixed-level Descriptions for This 16-QAM modulator (1/2)




```
module qam_mod( mout, din, clk);
    inout mout, din, clk;
    electrical mout, din, clk;
    parameter real fc = 100.0e6;
    electrical di1,di2, dq1, dq2;
    electrical ai, aq;

    serin_parout sipo( di1, di2, dq1, dq2, din, clk);
    d2a d2ai(ai, di1, di2, clk);
    d2a d2aq(aq, dq1, dq2, clk);
```

Structural descriptions

Verilog-A Mixed-level Descriptions for This 16-QAM modulator (2/2)




```
`define  PI  3.14159
real  phase;

analog  begin
    phase = 2.0 * `PI * fc * $realtime() + `PI / 4;
   V(mout) <+ 0.5 * (V(ai) * cos(phase) + V(aq) * sin (phase));
end
endmodule
```

Behavioral description of the QAM modulation

Analog System Simulation

- The standard approach to analog circuit simulation involves
  - Formulate the differential-algebraic equations for the circuit
  - Applying implicit integration methods to the sequence of nonlinear algebraic equations
  - Iterative methods, such as Newton-Raphson, to reduce to a set of linear equations
  - Using sparse matrix techniques to solve the linear equations
- These equations are not input directly, but derived from each of the models interconnected in the netlist

Outline

- Introduction
- Analog system description
- Data types and declarations
- Behavioral descriptions in Verilog-A
- Math functions and environment parameters
- References

Module Declaration

- A module is a definition of component, which can be a sub-system or a user defined primitive
- Module definition can not include another module definition
- Module Declaration consists of module name declaration, interface declaration, parameter declaration, module body and end statement

Module *module_identifier*(input/output ports) ;
input/output ports declaration ;

module body

endmodule

- A module can be defined with no port

Interface Declaration

- Interface declaration consists of port direction declaration, and port type declaration
- Port direction includes **input**, **output** and **inout**,

input : signals on the port can only be referenced but can't be set

output: signals on the port can be set, but can not be used in expressions

inout : the declare the port is bi-directional , can be set and be used

```
Module gainbk ( out, pin) ;
output out ;
input pin ;
voltage out, pin ;
analog
        v(out) <+ 3.0 * v(pin) ;
endmodule
```


Module Instantiations

- Instantiation is to incorporate another module into a module definition.
- Syntax :

module_identifier [*parameter_assignment*] *instance_list ;*

parameter_assignment : **#(*order_parameter_list*) or**
#(*named_parameter_list*)

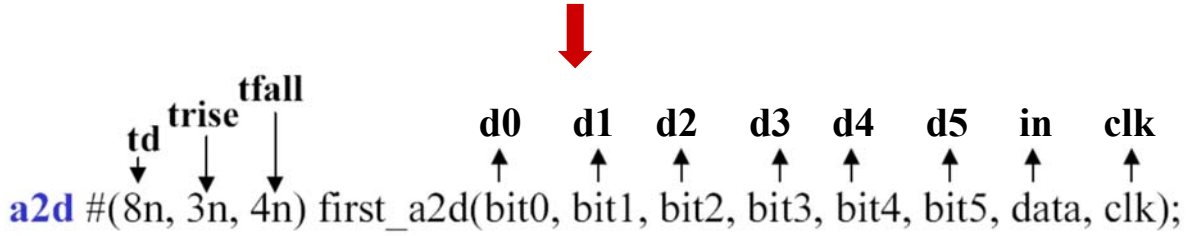
instance_list : ***name_module_instance*(|*list_of_module_connect*|)**

list_of_module_connect : ***ordered_port_connecttion* or**
named_port_connection

integrator #(1.0) **I1**(.out(nn1), .in(nn0)) ; or
integrator #(.gain(2.0)) **I2**(nn3, nn4) ;

Parameter Assigned by Order

```
module a2d(d0, d1, d2, d3, d4, d5,in, clk) ;
        input in, clk ;
        output d0, d1, d2, d3, d4, d5 ;
        electrical in, clk, d0, d1, d2, d3, d4, d5 ;
        parameter real td = 10n ;
        parameter real trise=4n ;
        parameter real tfall = 5n ;
```



- The assigned parameter number can be less or equal to the number of parameters defined in module, however, the parameter order must be kept.

Parameter Assigned by Name

```
module a2d(d0, d1, in, clk) ;
        input in, clk ;
        output d0, d1 ;
        electrical in, clk, d0, d1 ;
        parameter real td = 10n ;
        parameter real trise=4n ;
        parameter real tfall = 5n ;
```

```
a2d #(.trise(8n), .td(8n)) second_a2d(.d1(bit1),.in(data), .d2(bit2), . clk(clk));
```

- The assigned parameter are mapped by name, and only those modified parameters need to be included.
- Parameter name is proceeded by a period (.), and content must be enclosed by ( )

Data Types and Declarations

- Supports **integer**, **real**, and **parameter** data types in Verilog
- Supports parameter data types with range specification
- Supports **array** of real
- Supports new data type as a **node** which is for analog signal
- A node is defined with **discipline** which is further specified with **natures** of **potential** and **flow** and some associated attributes

Declaration of Integer

- An Integer declaration declares one or more variables of type integer

  integer *var_name* {, *var_names*} ;
- An integer variable hold values ranging from -2^{31} to $2^{31}-1$
- A variable might be an array with a range

  integer *var_name*[*lower_limit* : *upper_limit*]
- The indices must be constant expressions and must evaluate to an integer(positive integer, negative integer or zero)
- The arithmetic operations produce 2's complement result

Declaration of Real

- An real declaration declares one or more variables of type real

  real *var_name* {, *var_names*} ;
- A real variable is stored as **64**-bit quantities as described by IEEE STD-754-1985
- A variable might be an array with a range

  real *var_name*[*lower_limit* **:** *upper_limit*]
- The indices must be constant expressions and must evaluate to an integer(positive integer, negative integer or zero)
- Both integer and real variables are initialized to zero(0) at the start of simulation

Declaration of Parameters

- Parameters represent constants, hence can not be changed at run time, however, can be modified at compilation
- A parameter can be integer, array of integers, real or array of reals, and the default value must be specified.

  parameter real *var_name* = value ;

  parameter integer width=32, size=16 ;

  parameter real poles[0:3]={1.0, 2.132, 4.277, 6.186} ;
- The value is converted into correct type

  parameter real size=10 ;

  The value of size will be 10.0

Parameter with Range

- A parameter declaration can contain permissible range.
- The range can be specified with **()** in which values are not included or **[]** in which values are included

```
parameter real neg_rail = -15 from [-50:0) ;  -50 <= neg_rail < 0
parameter integer pos_rail=15 from (0:49) ;  0 < pos_rail < 49
parameter real gain=100 from [10:2000] ;  10 <= gain <= 2000
```

- **inf** is used for infinity, **-inf** is used for negative infinity
- Values can be excluded from range

```
parameter real res=1.0 from (0:inf) exclude 10 exclude (30:40] ;
```

- The range specified is checked at the compile time and is not a runtime check

Declare Signals in Different Systems

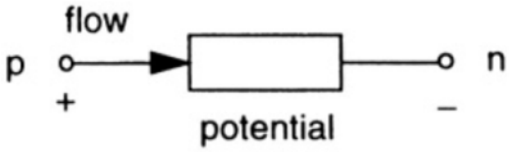
- Conservative Systems
  - Use of Kirchoff's laws
  - Electrical systems use KVL and KCL
  - Conservative systems use KPL and KFL
    - Applied to branches

KPL: Kirchoff's Potential Law
KFL: Kirchoff's Flow Law

- Signal Flow Systems
  - Only potential is associated with every node
  - Unidirectional
  - Notion of ports (input / output)
- A top-down design style must be able to combine both conservative and signal flow system
- A conservative port and signal flow port for the same discipline are compatible
  - They can be connected without errors

Conservative Systems

- Branches in conservative system
  - A path of flow between two nodes
  - Every branch has an associated potential and a flow



  - In conservative system, the charges or signals can enter a particular device **in both ways**

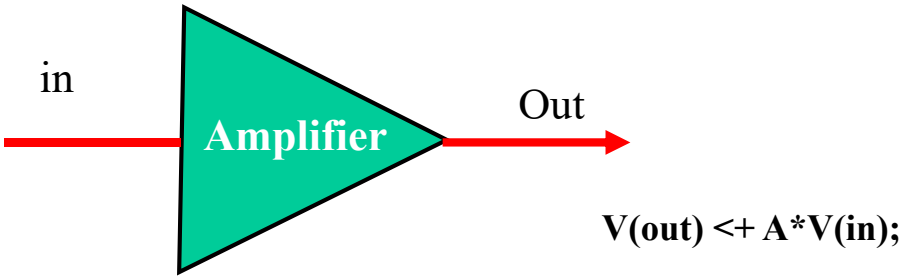
Declaration: electrical

- **In conservative systems:**

```
module resistor (a, b);
  inout a, b;
  electrical a, b; // access functions are V() and I()
  parameter real R = 1.0;

  analog
     V(a,b) <+ R * I(a,b);
endmodule
```

Both the ports are defined in the conservative discipline: **electrical**

Signal Flow Systems



In signal flow systems a signal can only enter a device in **one way only**

→ Change **in**: reflect as A*V(in) on the port **out**

Any change on **out**: not be seen by port **in**

Declaration: voltage, current

- **In signal-flow systems:**

(voltage amplifier)

```
module voltage_amplifier (in, out);
  input in;
  output out;
  voltage in, out;   //access function V()

  parameter real GAIN_V = 10.0;

  analog
      V(out) <+ GAIN_V * V(in);
endmodule
```

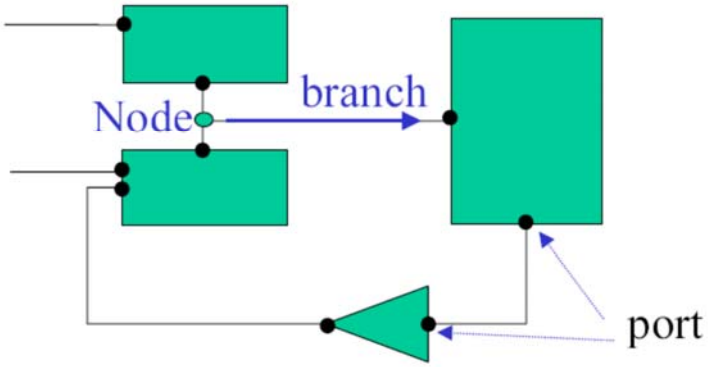
(current amplifier)

```
module current_amplifier (in, out);
  input in;
  output out;
  current in, out;   //access function I()

  parameter real GAIN_I = 10.0;

  analog
      I(out) <+ GAIN_I * I(in);
endmodule
```

Nets of signal flow disciplines may only be bound to **input** or **output** ports of the module, not to **inout** ports

Node and Branch

- A node is a connection point in the system
- A port is a component's(module's) external connection point, which is also a node
- A branch is a path of flow between two nodes
- There is no accumulation of flow on node



Node Declaration

Syntax : ***discipline*** [*range*] *list_of_nodes* ;

```
electrical  [MSB:LSB] nodea ;
voltage [6:0] n3, n4 ;
kinematic pump ;
magnetic inductor ;
```

(MSB and LSB are parameters)

- If a range is specified, the node is a vector node, or a analog bus
- For a conservative system, a node must have both potential and flow natures.

Node Access

- Nodes are always declared within interface of a module or in the module itself
- There is no local declaration of a node
  - That can not be declared inside block
- The contents of a node is accessed through access function defined in **nature**
- Ex:

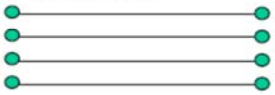
  V(*node1*) , where reference node 0 is used
  V(*node1*, *node2*)

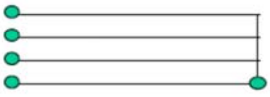
Branch Declaration and Access

- Branches are useful when specifying distinct parallel paths

Syntax : **branch** *terminals name_of_branch* ;
branch (p,n) bout, (ps, ns) bin ;

Vector terminal





Scalar terminal

electrical [3:0] n1, n2 ;
branch (n1, n2) resa ;

electrical [3:0] n1 ;
electrical n2 ;
branch (n1, n2) resb ;

Branch access :

V(bout) or I(bin)

Declare named branches

more clear than V(n1,n2)

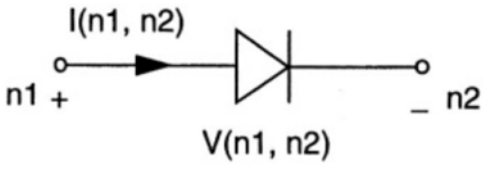
Outline

- Introduction
- Analog system description
- Data types and declarations
- Behavioral descriptions in Verilog-A
- Math functions and environment parameters
- References

Analog Model Properties

- The Verilog-A language can be used to represent different types of behaviors
  - Linear
  - Nonlinear
  - Piecewise linear
  - Integro-differential
  - Event-driven analog

Ex: non-linear diode model



$$i_d = i_{sat}(\exp(V(n1, n2)/V_T) - 1.0)$$

```
I(n1, n2) <+ isat*(exp(V(n1, n2)/$vt()) – 1.0);
```

$**vt** () is a Verilog-A system task that returns the thermal voltage

Statements for Behavioral Descriptions

- Analog statement
- Contribution Statements
- Procedural or Variable Assignments
- Conditional Statements and Expressions
- Multi-way Branching
- Iterative Statements

Analog statement

- Define the behaviors in terms of **contribution statements**, **control-flow**, and/or **analog event statements**
- All statements comprising the **analog** statement are evaluated at each point during an analysis
- The statement attached to an **analog** statement is usually a *block statement* delimited by a **begin-end** pair

```
analog begin
    <statements>
end
```

Contribution Statements

- Compute flow and potential values for the signals comprising the analog system

output_signal <+ *f*(input_signals);

- output_signal:
  - A branch potential or flow source
  - The target of the contribution operator (<+) assigned by the value of the right-hand side expression, f(input_signals)
- Ex: V(pout1, nout1) <+ expr1; I(pout2, nout2) <+ expr2;
  - expr1 and expr2 can be any expression of module signals, constants and parameters

Procedural or Variable Assignments

- The procedural assignments are used for modifying **integer** and **real** variables
- Similar to any programming language

```
real  x;
real  y[1:12];
analog begin
        …
        x = 5.0;
        y[i] = x;
        …
end
```

Left-hand side:
An integer or a real identifier or a component of an integer or real array

Right-hand side:
Any arbitrary expression constituted from legal operands and operators

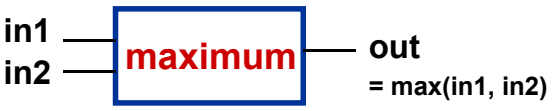
Conditional Statements and Expressions

- **if-else statement:**

```
if ( expr )
   <statement>
else
   <statement>
```

- The ternary operator (**?:**) can be used in place of the **if** statement when one of two values is to be selected for assignment

```
module maximum(out, in1, in2);
  inout out, in1, in2;
  electrical out, in1, in2;
  analog
    V(out) <+ ( (V(in1) > V(in2)) ? V(in1) : V(in2) );
endmodule
```

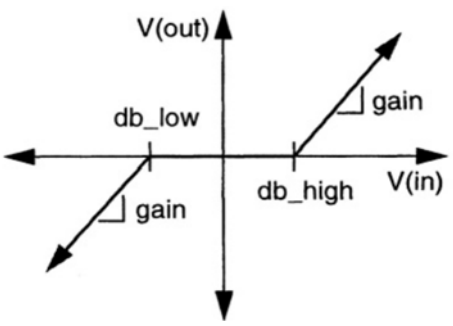


Multi-way Branching (1/2)

- **if-else-if** statement:

```
if ( expr1 )
   <statement1>
else if ( expr2 )
   <statement2>
else
   <statement3>
```

Ex: a dead-band amplifier




```
analog begin
   if ( V(in) >= db_high )
        vout = gain*( V(in) – db_high );
   else if ( V(in) <= db_low )
        vout = gain*( V(in) + db_low );
   else
        vout = 0.0;

    V(out) <+ vout;
end
```

Multi-way Branching (2/2)

- **case** statement: **case** (*expression*)
  test_expression
  { **,** *test_expression* } **:** *statement*
  ……
  default [:] *statements*
  endcase

```
case(x0)
    1:  x= 10 ;
    2:  x= 1 ;
    default: x=200 ;
endcase
V(n1) <+ x ;
```

```
real value ;

case (1)
   ( (value > 0)&&(value <= 1) ) :  $strobe("Category A");
   ( (value > 1)&&(value <= 2) ) :  $strobe("Category B");
   ( (value > 2)&&(value <= 3) ) :  $strobe("Category C");
   ( (value > 3)&&(value <= 4) ) :  $strobe("Category D");
   value <= 0 , value >= 4        :  $strobe("Out of range");

   default  $strobe("Error. Should never get here.");
endcase
```

Note: $strobe - display information on the screen

Iterative Statements: **repeat**

repeat (loop_cnt_expr)
    <statement>

- **repeat** executes <statement> a fixed number of times
- Evaluation of the constant loop_cnt_expr decides how many times a statement is executed

Ex: repeats the loop exactly 10 times while summing the first 10 digits

```
integer i, total ;
 i = 0 ;    total = 0 ;
 repeat(10)  begin
      i = i + 1 ;
      total = total + i ;
 end
```

Iterative Statements: **while**

while (loop_test_expr)
<statement>

- **while** executes a <statement> until the loop_test_expr becomes false
- If the loop_test_expr starts out false, the <statement> is not executed at all

Ex: counts the number of random numbers generated before **rand** becomes zero

```
integer  rand, count ;
rand = abs($random % 10) ;    count = 0 ;
  while(rand)  begin
    count = count + 1 ;
    rand = abs($random % 10) ;
  end ;
```


Iterative Statements: **for**

for (init_expr ; loop_test_expr ; post_expr)
<statement>

- Execute init_expr, or an assignment which is normally used to initialize an integer that controls the number of times the <statement> is executed
- Evaluate loop_test_expr
  - if the result is zero, the for-loop exits, and if it is not zero, the for-loop executes the associated <statement>
- Execute post_expr, or an assignment normally used to update the value of the loop-control variable, then continue

Ex: sum the first 10 even numbers

```
integer j,  total ;
total = 0 ;
for( j = 2; j < 22; j = j + 2 )
   total = total + j ;
```

Analog Operators

- The Verilog-A language defines analog operators for
  - Time Derivative
  - Time Integral
  - Linear time delay
  - Discrete waveform filters
  - Laplace Transform filters
  - Z-transform filters

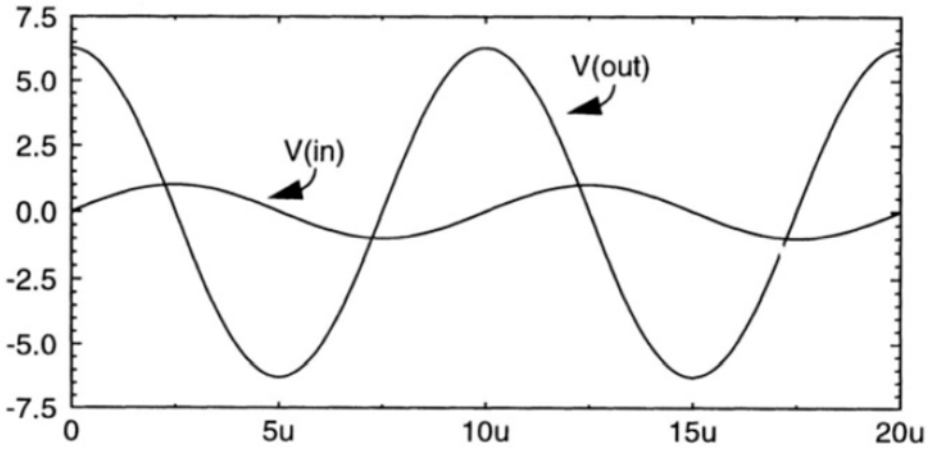
Time Derivative Operator

Operator	Comments
ddt(*expr*)	Returns $\frac{d}{dt}x(t)$, the time-derivative of *x, where x is expr.*
ddt(*expr; abstol*)	Same as above, except absolute tolerance is specified explicitly.
ddt(*expr; nature*)	Same as above, except nature is specified explicitly.

- In DC analysis the **ddt** operator returns a zero
- *abstol* is used as an absolute tolerance if needed
- *abstol* or derived from *nature*, applies to the output of the **ddt** operator and is the largest signal level that is considered negligible

ddt operator example

```
module ddt_op(out, in);
   inout out, in;
   electrical out, in;
   parameter real scale = 1.0e-6;

   analog
      V(out) <+ scale * ddt( V(in) );
endmodule
```



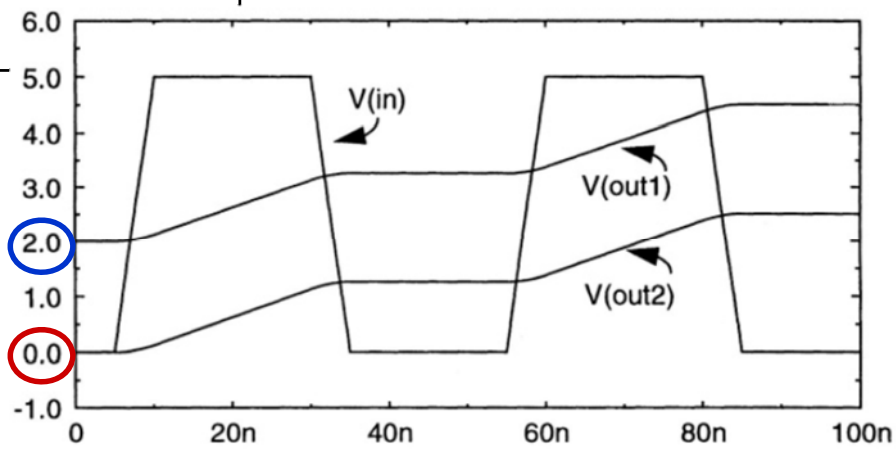
Time Integral Operator

Operator	Comments
`idt`(*expr*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where $x(\tau)$ is the value of *expr* at time τ, t_0 is the start time of the simulation, t is the current time, and c is the initial starting point as determined by the simulator and is generally the DC value (the value that makes expr equal to zero).
`idt`(*expr,ic*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where in this case c is the value of *ic* at t_0.
`idt`(*expr,ic,assert*)	Returns $\int_{t_a}^{t} x(\tau)d\tau + c$, where c is the value of *ic* at t_a, which is the time when *assert* was last nonzero or t_0 if *assert* was never nonzero.

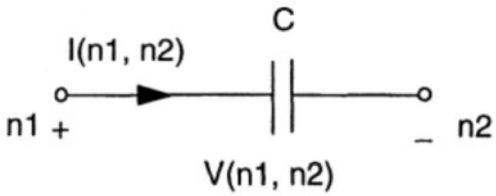
- When specified with initial conditions the **idt** operator returns the value of the initial condition in DC
- Without initial conditions , **idt** multiplies its argument by infinity in DC analysis

idt operator example

```
module idt_op(out1, out2, in);
   inout out1, out2, in;
   electrical out1, out2, in;
   parameter real scale = 1.0e6;
analog begin
   V(out1) <+ idt( scale*V(in), 0.0 );
   V(out2) <+ idt( scale*V(in), 2.0 );
end
endmodule
```



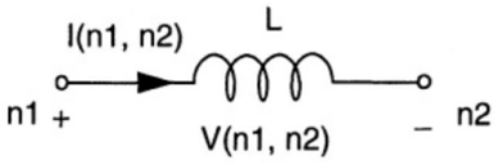
Behavioral Description: LC Case



$$i_c = \frac{d}{dt}(C \cdot V(n1, n2))$$

I(n1, n2) <+ **ddt**(C * V(n1, n2));

declared parameter

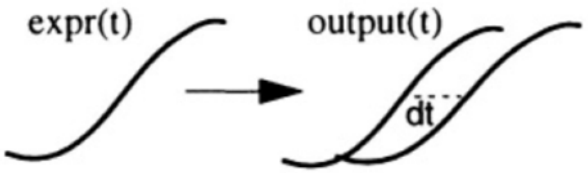


$$i_l = \int_0^t \frac{1}{L}(V(n1, n2))\,dt$$

I(n1, n2) <+ **idt**(V(n1, n2) / L);

Absolute Delay Operator

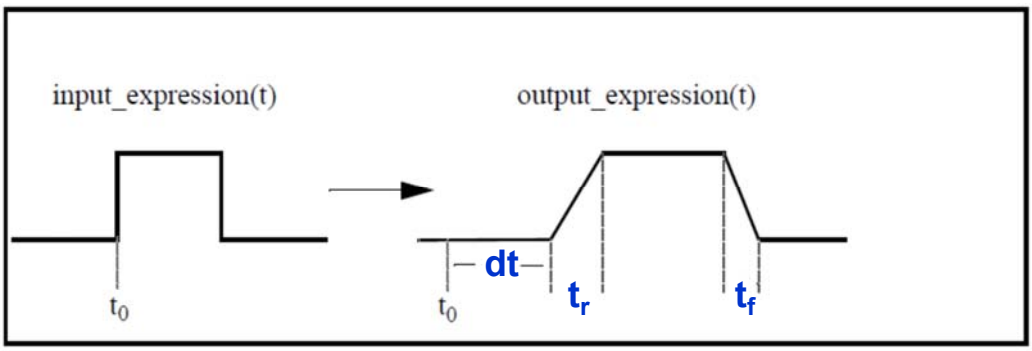
- Delay operator implements a transport or linear time delay for continuous waveforms
  - absdelay(expr , dt [, max_dt])



- The parameter dt must be nonnegative
- The effect of the delay operator in the time domain is to provide a direct time translation of the input

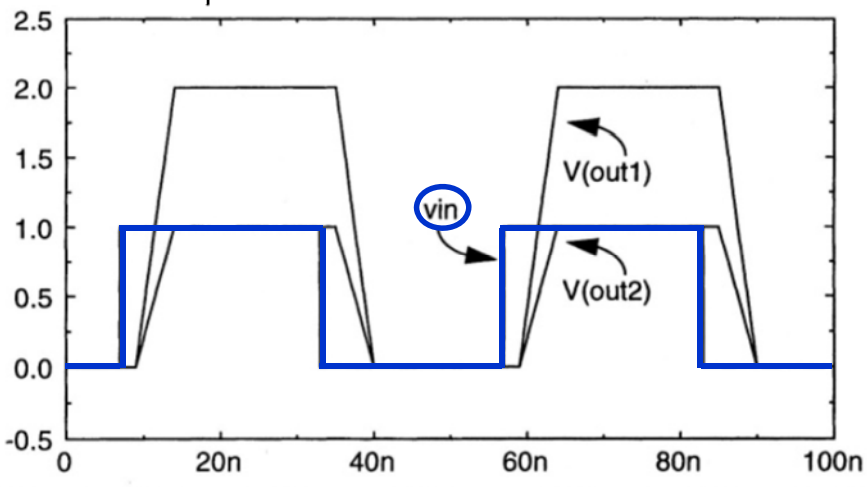
Transition Operator

- The transition operator smoothes out piece-wise constant waveforms.
- The transition filter is used to imitate transitions and delays on discrete signals
  - transition (expression, dt, tr, tf)
- The input expression to the transition operator must be defined in terms of discrete states.



transition operator example

```
…
analog begin
   if (V(in) > 0.5)
            vin = 1.0;
   else
            vin = 0.0;
   V(out1) <+ transition(vin, 2n, 5n, 5n);
   V(out2) <+ transition(2*vin, 2n, 5n, 5n);
end
endmodule
```



Slew Operator

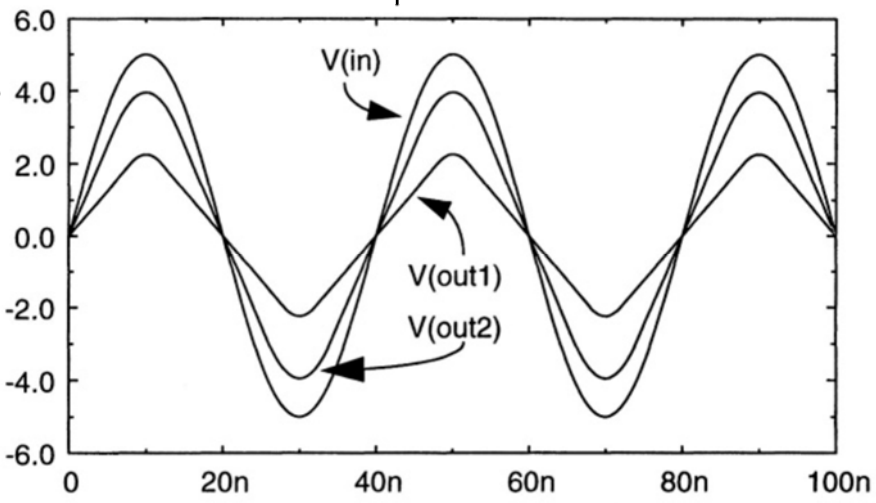
- The slew operator bounds the slope of the waveform
- used to generate continuous signals from piece-wise continuous signals
  - slew (expression, mpsr, mnsr)
    - mpsr : maximum positive slew rate
    - mnsr : minimum negative slew rate
- mpsr is a positive real number
- mnsr is a negative real number
- if only one rate is specified, the absolute value will be used for both rates

Slew Operator

- If no rate is specified, the slew operator passes the signal through without changing
- In DC analyses, the slew operator passes the value of the destination to its output
- In AC small-signal analyses the slew function has unity transfer function
  - except when slewing, in that case it has zero transmission through the slew operator

slew operator example

```
module slew_op(out1, out2, in);
   inout out1, out2, in;
   electrical outl, out2, in;
   analog begin
         V(out1) <+ slew(V(in), 0.5e9, -0.5e9);
         V(out2) <+ slew(V(in), 1e9, -1e9);
   end
endmodule
```



Laplace Transform Operators

- The Laplace transform operators implement lumped, continuous-time filters
  - laplace_zp (expression, numerator, denominator)
  - laplace_zd (expression, numerator, denominator)
  - laplace_np (expression, numerator, denominator)
  - laplace_nd (expression, numerator, denominator)
- The Laplace transform analog operator take vector arguments that specify the coefficients of the filter

laplace_zp: Zero-Pole Laplace Transforms

$$H(s) = \frac{\prod_{k=0}^{M-1} \{ 1 - \frac{s}{z_k^r + z_k^i} \}}{\prod_{k=0}^{N-1} \{ 1 - \frac{s}{p_k^r + p_k^i} \}}$$

$$H(s) = \frac{1 - s}{1 + s}$$

→ V(out) <+ laplace_zp(V(in), [1,0], [-1,0])

laplace_zd: Zero-Denominator Laplace Transforms

$$H(s) = \frac{\prod_{k=0}^{M-1} \{ 1 - \frac{s}{z_k^r + z_k^i} \}}{\sum_{k=0}^{N-1} \{ d_k s^k \}}$$

$$H(s) = \frac{1 - s}{1 + s}$$

→ V(out) <+ laplace_zd(V(in), [1,0], [1,1])

laplace_np: Numerator-Pole Laplace Transforms

$$H(s) = \frac{\sum_{k=0}^{M-1} \{ n_k s^k \}}{\prod_{k=0}^{N-1} \{ 1 - \frac{s}{p_k^r + p_k^i} \}}$$

$$H(s) = \frac{1 - s}{1 + s}$$

→ V(out) <+ laplace_np(V(in), [1,-1], [-1,0])

laplace_nd: Numerator-Denominator Laplace Transforms

$$H(s) = \frac{\sum_{k=0}^{M-1} \{ n_k s^k \}}{\sum_{k=0}^{N-1} \{ d_k s^k \}}$$

$$H(s) = \frac{1 - s}{1 + s}$$

→ V(out) <+ laplace_nd(V(in), [1,-1], [1,1])

Example: Butterworth Low-Pass Filter

$$H(s) = \frac{1}{s^5 + 3.236s^4 + 5.236s^3 + 5.236s^2 + 3.236s + 1}$$

```
module  laplace_op(out , in);
   inout out, in;
   electrical out, in;

  analog
        V(out) <+ laplace_nd (  V(in), [1],
                    [1,  3.236,  5.236,  5.236,  3.236,  1]);
endmodule
```

Z-Transform Operators

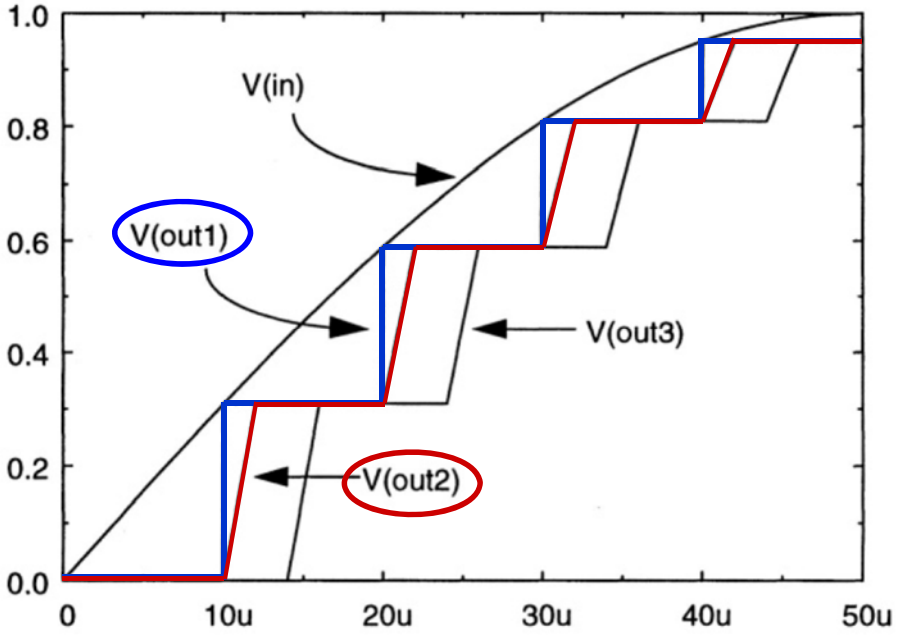
- The Z-Transform operators implement linear discrete-time filters
  - zi_zp(expression, numerator, denominator, T, trf ,t0)
  - zi_zd(expression, numerator, denominator, T, trf ,t0)
  - zi_np(expression, numerator, denominator, T, trf ,t0)
  - zi_nd(expression, numerator, denominator, T, trf ,t0)
    - T: the period of filter
    - trf: the transition time
    - t0: the initial delay time

Z-Transform Operators

- **T** : specifies the period of the filter
  - mandatory and must be positive
- **trf** : specifies the optional transition time and must be positive
  - if trf is zero, then the output is abruptly discontinuous
  - A Z-transform filter with zero transition time assigned directly to a source branch can generate discontinuities
- **t0** : specifies the time of the first transition and is optional
  - if t0 is not given, the transition occurs at t=0

Z-Transform Example

```
V(out1) <+ zi_nd(V(in), [ 1.0 ], [ 1.0 ], 10u );
V(out2) <+ zi_nd(V(in), [ 1.0 ], [ 1.0 ], 10u, 2u );
V(out3) <+ zi_nd(V(in), [ 1.0 ], [ 1.0 ], 10u, 2u, 4u );
```



Global Events

Global events are generated by simulator during simulation, and can not be generated by user model. These events are detected by system pre-defined name.

init_step(*analysis_function*)

The event is generated on the first time step in an analysis

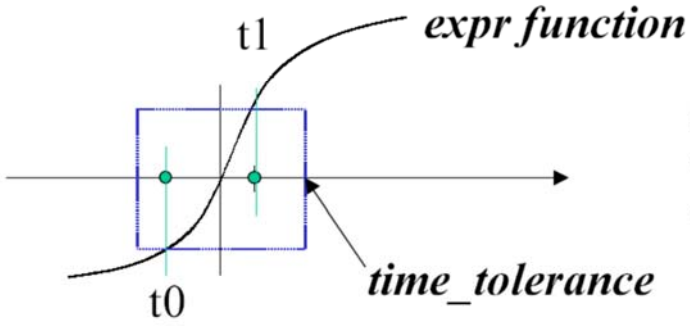
final_step(*analysis_function*)

The event is generated on the last time step in an analysis

Cross Event

- General From

  cross(*expr, direction, time-tolerance, expr-tolerance*)
- Generate event when ***expr*** crosses 0 in specified ***direction***
- Timepoint is placed just after the crossing within ***tolerance***
- To know the exact time of crossing, use **last_crossing()**
- **No cross function in loop and inside a function**

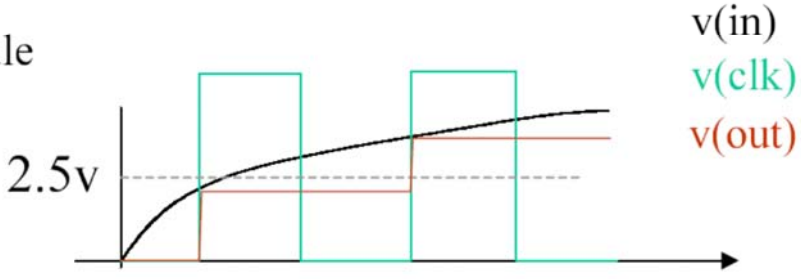


direction 0 : all zero crossing
direction 1 : zero increasing
direction -1 : zero descreasing

Cross Event Example

- Sample and hold

```
module sah(out, in, clk) ;
  output out ;
  input in, clk ;
  electrical out, in, clk ;
  real hold = 0 ;
  analog begin
     @( cross(V(clk) -2.5, +1, 0.01n))  hold = v(in) ;
     V(out) <+ transition(hold, 0, 10n ) ;
  end
endmodule
```



Timer Event

- Generate a timer event at specified times during simulation
- General From

  timer(***start_time*** [, ***period***] **)**
- If ***period*** is omitted, the event is generated once

```
module squarewave(out) ;
output out ;
electrical out ;
parameter real period=1.0 ;
integer x ;
analog begin
        @(initial_step) x=1 ;
        @(timer(0, period/2)) x = -x ;
        V(out) <+ transition(x, 0.0, period/100.0) ;
end
endmodule
```


Outline

- Introduction
- Analog system description
- Data types and declarations
- Behavioral descriptions in Verilog-A
- Math functions and environment parameters
- References

Basic Operators

Operator	Description
+ - * /	arithmetic
%	modulus
> >= < <=	relational
==	logical equality
!=	logical inequality
!	logical negation
&&	logical AND
||	logical OR
~	bit-wise negation
&	bit-wise and
|	bit-wise OR
^	bit-wise XOR
^~ or ~^	bit-wise equivalence

Operator	Description
<<	left shift
>>	right shift
? :	condition
or	event OR

Built-in Mathematical Functions

Function name	Description	Domain(Range)
ln(x)	Natural logarithm	x > 0
log(x)	Decimal logarithm	x > 0
exp(x)	Exponential	x < 80
sqrt(x)	Squart root	x >= 0
min(x,y)	Minimum	All x, All y
max(x,y)	Maximum	All x, All y
abs(x)	Absolute	All x
pow(x,y)	Power, x^y	All x, All y
floor(x)	Floor function	All x
ceil(x)	Ceiling function	All x

Transcendental Functions

Function name	Description	Domain(Range)
sin(x)	Sine function	All x
cos(x)	Cosine function	All x
tan(x)	Tangent function	x <> n(π / 2) , n is odd
asin(x)	Arc-sine function	-1 <= x <= 1
acos(x)	Arc-cosine function	-1 <= x <= 1
atan(x)	Arc-tangent function	All x
atan2(x,y)	Arc-tangent of x/y	All x, All y
hypot(x,y)	sqrt(x^2+y^2)	All x, All y
sinh(x)	Hyperbolic sine	All x
cosh(x)	Hyperbolic	All x
tanh(x)	Hyperbolic	All x
asinh(x)	Arc-hyperbolic sine	All x
acosh(x)	Arc-hyperbolic cosine	x >= 1
atanh(x)	Arc-hyperbolic tangent	-1 <= x <= 1

Environment Parameters

Function	Returns
$realtime or $realtime()	Current simulation time in seconds
$temperature	Ambient temperature in kelvin
$vt	Thermal voltage (kT/q)
$vt(temp)	Thermal voltage at given temperature

References

- Designer's Guide
  `http://www.designers-guide.org/`
  - Forum
  - Verilog-A model library (VBIC, MOS11, JFET, etc.)
- MCAST (Prof. CJ Richard Shi)
  `http://www.ee.washington.edu/research/mscad/shi/mcast.html`
  - Automatic compiler beats hand-coded C
- D. FitzPatrick, I. Miller, Analog Behavioral Modeling with the Verilog-A Language, Springer,1998.
- Cadence Verilog-A Language Reference, Version 6.1, Dec. 2006.
- Analog Hardware Description Language Verilog-A Training Manual, 2002.
- Mixed-Signal IC Design Kit Training Manual, 2002.
- G. Coram, "Verilog-A: An Introduction for Compact Modelers," MOS-AK/ESSDERC/ESSCIRC Workshop, 2006.

Examples

- Verilog-A model library at
  `http://www.designers-guide.org/VerilogAMS/`
  - VBIC, MOS11, JFET, etc.
- Silvaco "public domain" models (non-commercial use)
  `https://src.silvaco.com/ResourceCenter/en/downloads/verilogA.jsp`
  - BSIM3, BSIM4, BJT, etc.