ECE1388 – TUT05a – Verilog-A Language

Analog behavioral modeling can help speed up verifications for larger, complex circuits where simulations are much longer. This tutorial is an introduction to analog behavioral modeling using Verilog-A running in Spectre. It gives examples to help you understand the basic modeling concepts. Most of the content is derived and summarized from [1].

Introduction

Analog Behavioral Modeling deals with creating and simulating models based on a desired external circuit behavior. Models are best used to represent circuit block behavior and not simply replicate individual transistor characteristics. Models can be as complex as necessary. Often, initial behavioral models need to carry only the basic properties, such as an operational amplifier might have voltage swings, impedances, and gain. In other cases, there might be a need to model slew rates, differential signals and bandwidth properties. There are six main reasons to consider modeling:

• Design exploration

• Verify connectivity

• Verify functionality

• Speed up simulations

• Reuse in future designs

• To create a portable design IP

Analog behavioral modeling is part of a wider design methodology called "top-down design". This may seem obvious, but there are a number of aspects that require careful consideration to take full advantage of modeling. Top-down design starts with creating a hierarchical design. This is a common design practice today, especially for large designs. However, the key is to make all circuit blocks in the hierarchy pin-to-pin compatible so that each can be represented by either a model or an actual transistor-level circuit block. Later, the views can be toggled between model and transistor for mixed-level simulation.

Verilog-A Overview

Verilog-A was derived from Verilog HDL in 1996 by the Open Verilog International (OVI) organization, and was later extended to Verilog-AMS. Verilog-AMS is a superset of Verilog-HDL and Verilog-A and a true mixed-language, where both are written into a model. Many of the Verilog-A constructs are the same in Verilog-AMS, with minimal differences. Verilog-HDL in Verilog-AMS is extended to support both Verilog-A and Verilog-AMS connections.

Language basics:

Verilog-A has the ability to model a variety of disciplines, the most common of which are *electrical*, *magnetic*, *thermal*, *kinematic*, and *rotational*. You can also define your own disciplines. For the most part, the electrical discipline, which is expressed as voltages and currents, is used primarily for integrated circuit modeling. Along with disciplines, there are three basic modeling styles:

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

- **Conservative** includes both potential and flow. For an electrical system, these translate to voltage and current respectively.
- **Signal-Flow** includes only potential. Useful for high-level modeling such as electrical systems that you initially don't want to include current definitions.
- **Event** driven models which evaluate events. This can be useful in modeling digital, mixed-signal and high-level systems.

```
Nature Voltage
  abstol = 1u;
  units = "V";
  access = V;
 huge = 1e5;
endnature
```

Using Kirchhoff's law, *Nodes* are defined as being where branches interconnect and branches are the paths between nodes. Disciplines are described by *Natures,* which describe the tolerance (abstol), evaluated units (units), and name (access). Usually, Disciplines and Natures are described in a file called disciplines.h (disciplines.vams for Verilog-AMS) which is included during netlisting. These can also be included in the actual model file. A constants.h (constants.vams for Verilog-AMS) file which carries commonly used mathematical and physical constants can also be included. Mathematical constants have an `M_ prefix and physical constant `P_ prefix (examples: `M_TWO_PI = 2 π and `P_Q = Q).

```
include "disciplines.h"
include "constants.h"

module rlc (a,b);
  electrical a,b;
  parameter R=1 exclude 0;
  parameter C=1;
  parameter L=1 exclude 0;
 branch (a,b) res, cap, ind;

 analog begin
   I(a,b) <+ idt(V(a,b))/L;
 V(res) <+ R*I(res);
 I(cap ) <+ C*ddt(V(cap));
end

endmodule;
```

Verilog-A modules have pin connections (called ports) and behave like any component in a circuit, such as a transistor or resistor. The syntax of the model file, except the inclusion of disciplines.h and constant.h files, starts with a module declaration which carries the module name and declared pin names. The pin connections have declared port directions and disciplines. In IC design, the discipline will likely be electrical, but there are cases where only a voltage or current will be used. In other cases, cross-discipline models can be described where electrical is coupled to magnetic and rotational disciplines. Parameter declarations optionally allow the parameter to be passed from module to schematic without editing the model file. Where components share the same nodes and are often referenced, a branch statement can provide a name to each branch. If there are variables used in the model, such as real and integers, these need to be declared before use. Variable names must start with a letter or _, and are case sensitive.

- Analog begin – end**:** Encloses the analog behavior of the module. Often, this section is where voltage and currents from outside pins are sensed, checked for signal crossings, mathematically conditioned, then pushed back out to the circuit or stored in a file.
- The simulator interprets the model in sequential steps. For example, at each timestep for a transient, each relationship is evaluated and can depend on previous lines.
    - Inside a controlled loop, such as conditional expressions, all expressions are evaluated together and dependent on outside the loop.
- Contribution Operator (<+) is a line in the model that passes conditioned signals back to the rest of the circuit being simulated. This can be additive, such that there could be multiple expressions passing signals to the same outside pin.
- All lines in the model file end with a ";" except basically, *begin, else* and *end* statements.

Mathematical Functions and Operators:

Standard mathematical functions (standard math, logarithm, trigonometry), random numbers (uniform, Gaussian, exponential), analog operators (derivative, analog delay) and analog filters (slew, Laplace). In the following tables, you can find the built-in Math functions and Verilog-A operators.

Verilog-A Built-In Math Functions

Function	Description	Domain
ln(x)	natural log	x>0
log(x)	decimal log	x>0
exp(x)	exponential	x<80
sqrt(x)	square 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	if x>=0, all y if x<0, int(y)
floor(x)	floor	all x
ceil(x)	ceiling	all x
sin(x)	sine	all x
cos(x)	cosine	all x
tan(x)	tangent	x !=n(π/2), n is odd
asin(x)	arc-sine	-1 <=x<=1
acos(x)	arc-cosine	-1 <=x<=1
atan(x)	arc-tangent	all x
atan2(x,y)	arc-tangent x/y	all x, y, except 0
hypot(x)	sqrt($x^2 + y^2$)	all x, y
sinh(x)	hyperbolic sin	all x
cosh(x)	hyperbolic cos	all x
tanh(x)	hyperbolic tan	all x
asinh(x)	a-hyperbolic sin	all x
acosh(x)	a-hyperbolic cos	x=>1
atanh(x)	a-hyperbolic tan	-1<=x<=1

Verilog-A Operators

+	plus
-	minus
*	multiply
/	divide
%	modulus
<	less than
>	greater than
<=	less than, equal to
>=	greater than, equal to
!==	case inequality
===	case equality
!=	logical not equal
(?:)	ternary
==	logical equal
!	logical negation
&&	logical and
||	logical or
~	bit negation
&	bit and
|	bit or
^	bit xor
^~,~^	bit equivalence
<<	left shift
>>	right shift
or	event or

Analog Operators:

Analog operators cannot be used in *functions, repeat*, *while*, or *for* statements. If *if* or *case* statements are used, the controlling expression must consist entirely of literal numerical constants, parameters, or the analysis function. The following is a list of analog operators:

- **ddt(x):** is the differentiator operator and takes the time derivative of it's argument.
- **idt(x):** is the integrator operator and operates the time integral of its argument.
- **idtmod(x):** is the circular integrator and applies a modulus operation on the time integral of its operator. Used for periodic operation.
    - *Syntax:* **idtmod**(integrand *x*, initial condition y_0, modulus *m*, offset *b*, tolerance)
- **absdelay(x):** time delay for signal x.
- **last_crossing(x):** is the operator returning the time of last crossing of it's argument.
- **transition(input_sginal, time_delay, risetime, falltime):** is one of the commonly used operators. It adds delay, rise time and fall time to the signal.
    - For smoothly varying signals, use slew filter instead.
- **slew(input_signal, slew_pos, slew_neg):** is the slew filter and bounds the signal rate-of-change at the output.

- **Laplace filters:** are the linear continuous-time filter functions with fixed poles and zeros.
- **Z filters:** are the linear discrete-time filters with fixed period, poles and zeros.

Here are two simple examples for a VCO and a DFF with some of the analog operators discussed above:

Example: Basic Sinusoidal VCO

```
module vco(out,in);
  voltage out,in;
  parameter real k = 1M;
  real phase, freq;

  analog begin
    freq = k*V(in);
    phase = idtmod(freq,0,1);
    V(out) <+ cos(2*`M_PI*phase);
   $bound_step(1/(10*freq));
  end
endmodule
```

Example: Analog D Flip-Flop

```
module dff (q,d,clk);
  voltage q, d, clk;
   input clk, d;
   output q;
   parameter real td=0 from [0:inf], tr=0 from [0:inf];
   parameter integer dir=1 from [-1:1] exclude 0;
   parameter real Vdd=5 from (0:inf);

   integer state;

   analog begin
     @cross(V(clk) – Vdd/2, dir)
        state = (V(d) > Vdd/2);
     V(q) <+ transition(state*Vdd,td,tr);
   end

endmodule
```

- **bound_step function:** is a simulator directive.
  - Example: ***$bound_step****(1/step/frequency)*
    - Must be used to notify simulator to take time step no larger than its argument, so that simulator can successfully update the smooth change in sin wave outputs. In this case, the simulator changes the output according to the specified time step even if there's no change in the input.

Analog event driven modeling:

- **@(event):** used for event driven models, such as edge triggered circuits. In the following table summarizes the most common event commands.

Analog Event Types	Description
cross(expr,dir)	At analog signal crossings
above(expr)	At signal low-to-high crossing, and when above at DC
timer(time,dt)	Periodically or at specific times
Initial_step	At the beginning of simulation
final_step	At the end of the simulation

  - For example cross event command:
    - Syntax: ***cross(expr, direction, timeTol, exprTol)***
    - Generates event when expr crosses 0 in a specified direction
    - Timepoint is placed just after the crossing, within tolerances
    - To know the exact time of crossing, use last_crossing(expr)

Looping and Conditional Statements:

- **If-else:** The If-else is a binary conditional set of statements under control of specified conditional expressions.
    - Syntax: **if** (expression1) statement1;
      else if (expression2) statement2;
      else statement3;
- **Case:** the case expression controls of a series of statements to run depending on what the expression is equal to.
    - Syntax: **case** (expression)
      value1: statement1;
      value2: statement2;
      value3: statement3;
      default: statement;
      endcase
- **Repeat:** the repeat loop statement runs for a fixed number of times as determined by the constant_value.
    - Syntax: **repeat** (constant_value) statement;
- **While:** the while loop statement is used when you want to leave the loop when an expression is no longer valid.
    - Syntax: **while** (expression) statement;
- **For:** the for loop statement runs a fixed number of time.
    - Syntax: **for** (initial_statement; expression; step_statement) statement;

Simulator Interface Functions:

Verilog-A can provide conditional controls, information commands, and small-signal stimulus functions. Here are a few of the functions:

- **analysis():** Analysis done on a condition basis
    - *Example:*

```
module cap1 (a,b);
electrical a,b;
parameter real c=0, ic=0;
analog begin
if (analysis ("ic"))        ←   excute on transient IC analysis only
V(a,b) <+ ic;
else                        ←   execute all other analyses
I(a,b,) <+ ddt(c*V(a,b));
end
endmodule
```

- **$discontinuity():** Used to make a model discontinuity at current point. A discontinuity(0) announces a discontinuity in a descriptive equation. A discontinuity(1) indicates a discontinuity in the first derivative (slope) of the equation.
- **$abstime, $temperature, $vt, $vt():** These are environment functions that provide information about the current simulation environment.
    - *Examples:*

```
therm_volt = `P_K * $temperature / (`P_Q * emis_coef); //ambient temperature in degees Kelvin
V(out) <+ sin (2 * `M_PI * freq * $abstime); // at current simulation time
$strobe("Simulation time = %e", $abstime); // at current simulation time
```

thermal_voltage = $vt*; // at current simulation temperature*
vt_temp = $vt(76)*; //thermal voltage at 76 degees Kelvin*

- **ac_stim(), white_noise(), flicker_noise(), noise_table():** Used for small-signal noise modeling.
  - *Example: I(diode) <+ white_noise(2 * `P_Q * I(diode), "source1");*
- **$bound_step():** describe earlier in the VCO example.

REFERENCES

I. Cadence Platform Application Note, "Creating Analog Behavioral Models – Verilog-AMS Analog Modeling" Feb 2003.

II. Verilog-A Language Reference Manual.