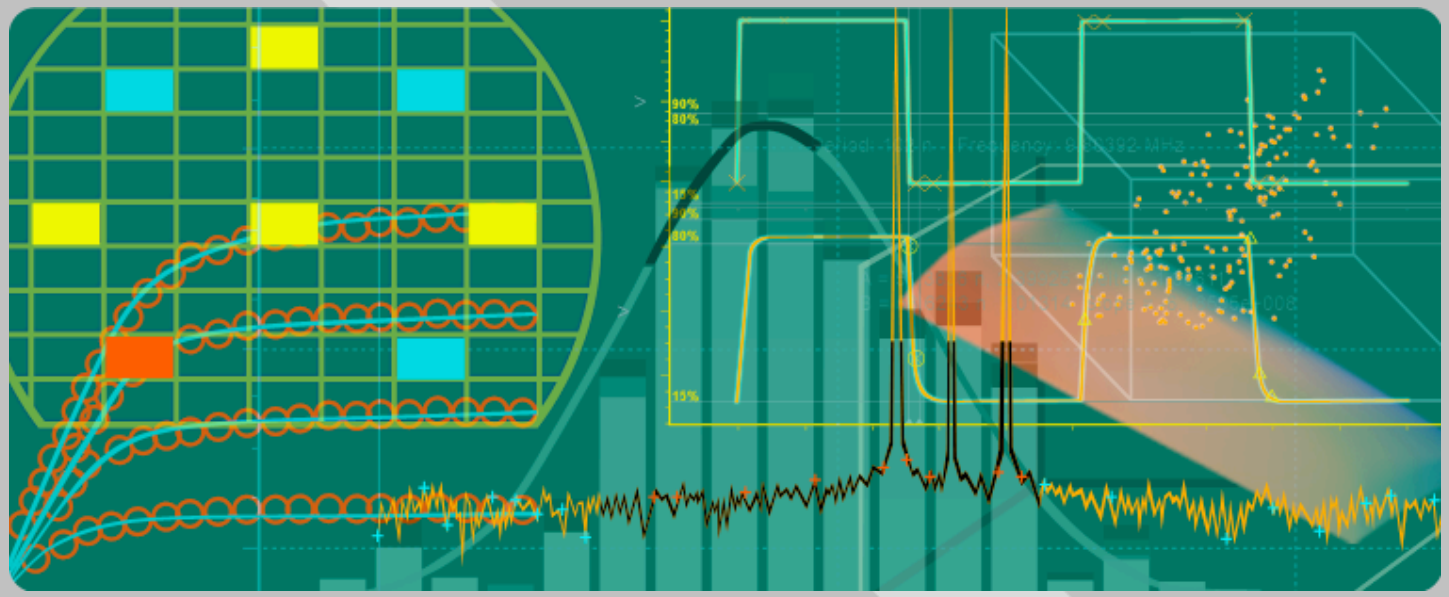
SmartSpice Training Program



Part 5: SmartSpice Verilog-A

SIMUCAD DESIGN AUTOMATION

What Is SmartSpice Verilog-A?

- The SmartSpice Verilog-A is a high-level Analog Hardware Description Language (AHDL) that uses modules as the basic component to describe the structure and behavior of analog systems and their components
- With the analog statements of VERILOG-A, you can describe a wide range of conservative systems and signal-flow systems; such as electrical, mechanical, fluid dynamic, and thermodynamic systems
- To simulate systems that contain VERILOG-A components, you must have a VERILOG-A license and the SmartSpice simulator installed on your system
- To describe a system, specify both the structure of the system and the behavior of its components
- In VERILOG-A with the SmartSpice circuit simulator, the structures are defined at different levels
- At the highest level, you can define the overall system structure in a netlist
- At the lower, more specific levels, you can define the internal structure of modules by defining the interconnections among submodules

Introduction to Verilog-A in SmartSpice

- Why use Verilog-A in SmartSpice?
  1. Trade off speed and accuracy during verification or top down design
  2. By using **`.ST`** to perform Verilog-A module parameters sweep
  3. Using **`.MODIF`** to perform optimization for Verilog-A parameters

- How to use Verilog-A in SmartSpice
  1. Create a **Verilog-A module in a source file with .va extension**
  2. Instantiate the Verilog-A modules in the netlist by using **`.YVLG`**
  3. Use the **`.verilog`** command in the input deck to include the verilog-A source file

Representing a System in Verilog-A

- A system is a collection of interconnected components that, when acted upon by a stimulus, produce a response
- A hierarchical system is a system in which the components are also systems
- A primitive component (leaf component) is a component that has no subcomponents
- Each primitive component connects to zero or more nets
- Each net connects to a signal which can traverse multiple levels of the hierarchy.
- The behavior of each component is defined in terms of the values of the nets to which it connects
- The VERILOG-A language allows analog and mixed-signal systems to be described by a set of components or modules
- A signal is a hierarchical collection of nets which, because of port connections, are contiguous
- The nets for a signal are in the discrete domain called a digital signal
- The nets that make up a signal are in the continuous domain; the signal is an analog signal
- The net that consists of signals from the continuous and discrete domain is called a mixed signal

The SmartSpice Verilog-A Simulation Flow

- The main stages of the VERILOG-A interface are described below:
  - **1. Compilation Phase:** During the sourcing of a SmartSpice netlist, VERILOG-A files referred to by the .verilog command cards are compiled by the VERILOG-A Simucad COMPILER. As a result of the compilation, a C file is produced for each module parsed.
  - **2. Linkage Phase:** The generated C files are then automatically parsed by the Simucad C-INTERPRETER, and transformed into an optimized pseudo-code that will be executed during the simulation. If the C files are compiled by a gcc or the native C compiler, a Dynamically Linkable Library (a .so file) is produced. This library is then linked to the SmartSpice executable.
  - **3. Simulation Phase:** Once all VERILOG-A modules have been incorporated, the simulation in SmartSpice is done as usual. The compilation, and the linkage are in fact transparent to the user. Simulating a netlist with VERILOG-A module is still done the same way as with a traditional netlist: first the netlist is sourced, then a simulation is run, and the results output in a text or graphical form.

Verilog-A System Types

- Two types of analog systems can be described with VERILOG-A:
  - Conservative systems
    - Includes those described by conventional spice, incorporates a set of constraints within the system that insures conservation of charges, fluxes and so forth, within the system
    - Have two values that are assigned to every node: a potential value and a flow value
    - The potential of a node is shared by all ports and nets connected to it
    - The flow of a node is such that the sum of all continuous nodes are equal to zero
    - For this reason, the conservation laws, such as Kirchoff 's Potential Law (KPL) and Kirchoff's Flow Law (KFL), can be applied to every node
    - KPL and KFL are generalizations of KVL and KCL for electrical systems which allow the conservation laws to be applied to any conservative system
  - Signal flow systems
    - Employ a different level of formulation, which focuses only on the propagation of signals throughout the system
    - Associate only a single value with each node. As a result, a signal-flow port must be unidirectional. If the component has two ports, one port is the input and the other one must be the output

Verilog-A Inverter Model

```
VERILOG-A for inverter (inv.va):
      `include "discipline.h"
    module inv(in, out);
      input in;
      output out;
      electrical in, out;
      parameter real vout_high = 5;
      parameter real vout_low = 0;
      parameter real vth = 2.5;
      parameter real tdelay = 5n;  //propagation delay
      parameter real trise = 8.5n; //rise time
      parameter real tfall = 10n;  //fall time
      real val;
      analog begin
          @( initial_step )
          begin // initial condition
          if ( V(in) > vth ) val = vout_low;
          else val = vout_high;
      end
      @( cross( V(in) - vth, +1) ) val = vout_low;  // Vin>vth => vout_low
      @( cross( V(in) - vth, -1) ) val = vout_high; // Vin<vth => vout_high
      V(out) <+ transition(val, tdelay, trise, tfall);
  end
Endmodule
```

Verilog-A Inverter SPICE Test Bench

SPICE test bench for inverter (inv.in):

```
*----LOGIC ( inverter ) Circuit-----*
*
.verilog "inv.va "
.options post probe iplot_one numdgt=10 accurate
v01 vcc 0 dc 5
v02 in 0 pulse(0 5 1u 1n 1n 1u 2u)
*
YVLG_NOT in out inv
.tran 1n 10u
.iplot v(in) 'v(out) + 10'
.probe all
.end
```

Verilog-A NAND Model

```
Verilog-A for NAND (nand.va):
     `include "discipline.h "
     module dnand(in, out);
        parameter real size = 2 from [2:inf),
                vout_high = 5,
                vout_low = 0 from (-inf:vout_high),
                vth = 2.5,
                tdelay = 5n from [0:inf),
                trise = 10n from [0:inf),
                tfall = 12n from [0:inf);
     input [0:size - 1] in;
     output out;
     voltage in, out;
     integer in_state[0:size - 1], i;
     integer out_state;
     real vout;
     analog begin
      @(initial_step)
               for (i = 0; i < size; i = i + 1) in_state[i]=0;
      generate i (0,size - 1) begin
           @(cross(V(in[i]) - vth)) begin
        In_state[i] = V(in[i]) > vth;
                  out_state = 1;
                  for (i = 0; i < size; i = i + 1)
                  if (!(out_state && in_state[i])) out_state = 0;
                  if (out_state) vout = vout_low; // inversion of output
                  else vout = vout_high;
           end
     end
     V(out) <+ transition(vout, tdelay, trise, tfall);
       end
endmodule
```

Verilog-A DFF Model

VERILOG-A for DFF (dff.va):

```
`include "discipline.h"
module vdff(q, qbar, clk, d);
  input clk, d;
  output q, qbar;
  voltage q, qbar, clk, d;
  parameter real tdelay = 5n from [0:inf),
                 ttransit = 5n from [0:inf),
                 vout_high = 5,
                 vout_low = 0 from (-inf:vout_high),
                 vth = 2.5;
  integer x;
  analog begin
    @(initial_step)
      x = 0;
    @(cross(V(clk) - vth, +1 )) x = (V(d) > vth);
    V(q)    <+ transition( vout_high *  x + vout_low * !x, tdelay, ttransit);
    V(qbar) <+ transition( vout_high * !x + vout_low *  x, tdelay, ttransit);
  end
endmodule
```

Verilog-A LPF

Verilog-A for LPF (lpf.va):

```
`include "discipline.h"
`include "constants.h"
module lpf1(in, out);
  input in;
  output out;
  voltage in, out;
  parameter real freq_p1 = 1M from (0:inf);
  analog
     V(out) <+ laplace_nd(V(in), {1} ,{1,1/(`M_TWO_PI*freq_p1)});
endmodule
```

Verilog-A HPF

Verilog-A for HPF (hpf.va)

```
`include "discipline.h"
`include "constants.h"
module hpf(in, out);
  input in;
  output out;
  voltage in, out;
  parameter real freq_p1 = 1M from (0:inf);
  analog
    V(out) <+ laplace_nd(V(in), {0, 1 / (`M_TWO_PI*freq_p1)},
      {1, 1 / (`M_TWO_PI*freq_p1)});
endmodule
```

Verilog-A Active Analog Circuits

- This section has analog circuit examples using VERILOG-A, although describing analog circuit behavior is difficult because the simple behavior is not the whole behavior of the analog circuit
- If the user wants to describe VERILOG-A code for an analog circuit, one must know the analog specification and characteristics and the intended use or purpose for the model

Verilog-A OPAMP

Verilog-A Source (amp.va):

```
      // simple opamp
      `include "descipline.h"
      module amp(in, out);
        input in;
        output out;
        electrical in, out;
        //parameter real gain = 10;
        parameter      avcc = 5;    //Analog supply voltage source
        parameter      avss = 0;    //Analog ground reference
        parameter      vref = 2.5;  //input reference voltage
        parameter real R1 = 1k;     //initial resistance
        parameter real R2 = 1k;     //initial resistance
        real A, gain, vout;
        analog begin
            @(initial_step) begin
              gain = R2 / R1;
              A = -1 * gain; //invert signal
            end
            vout = A * (V(in) - vref) + vref;
            if (vout > avcc)
              vout = avcc;
            if (vout < avss)
              vout = avss;
            V(out) <+ vout;
      end
    endmodule
```

Verilog-A Sample & Hold

Verilog-A Source (sample_hold.va):

```
`include "discipline.h"
module sample_hold(in, out, clk);
  input in, clk;
  output out;
  voltage in, out, clk;
  parameter real clk_vth = 2.5;
  real v;
  analog begin
     @(initial_step)
        v=V(in);
     if (analysis("static") || (V(clk) > clk_vth))
        v = V(in);                    // passing phase
     @(cross(V(clk) - clk_vth,0))
        v = V(in);                    // sampling phase
     V(out)<+ v;
  end
endmodule
```

Verilog-A DAC (Part I)

```
Verilog-A for DAC (dac.va)
//----- DAC -----//
`include "discipline.h"
`include "constants.h"
module dacn(in, out);
  parameter real fullscale = 5;              //supply voltage
  parameter integer maxbit = 10, bit = 10; //Resolution
  input [0:bit-1] in;
  output out;
  electrical out;
  electrical [0:bit-1] in;
  real vlump[maxbit:1];
  real vout[maxbit:1];                       //voltage
  real outv, vth;
  integer i;                                 //index loop
  integer code[bit-1:0];                     //digital code analog begin
  analog begin
    @(initial_step) begin
      vth = fullscale/2;
      for (i = 1; i <= maxbit; i = i + 1) begin
        vlump[i] = fullscale / pow(2,i);
      end
    end
```

Verilog-A DAC (Part II)

Verilog-A for DAC (dac.va) (cont'd)

```
    for (i = 1; i <= bit; i = i + 1) begin
      if (V(in[i-1]) > vth) begin
        code[i-1] = 1;
      end else begin
        code[i-1] = 0;
      end
      vout[i] = vlump[i] * code[i-1];
    end
    if (bit < maxbit) begin
      for (i = maxbit; i > bit; i = i - 1) begin
        vout[i] = 0;
      end
    end
    outv =
   vout[1]+vout[2]+vout[3]+vout[4]+vout[5]+vout[6]+vout[7]+vout[9]+vout[10];
    // V(out) <+ transition(outv, 50n, 50n, 50n);
    V(out) <+ outv;
  end
endmodule
```

Verilog-A ADC (Part I)

```
Verilog-A ADC (part I)
Verilog-A for ADC (adc.va)
//----- Pipelined ADC-----//
`include "discipline.h"
`include "constants.h"
module adc (in, out, clk);
  parameter integer bit = 10;      // ADC resolution
  parameter real fullscale = 5.0, //supply voltage
               vth = 2.5,         //threshold
               dly = 10n,         // transition delay
               ttime = 1n;        // transition rising time
  input in;                       // input analog voltage
  input clk;                      // input clock
  output [bit-1:0] out;           // digital vector output
  electrical in, clk;
  electrical [bit-1:0] out;
  real sample;
  integer result[bit-1:0];        // integer array
  integer i;                      // index loop
```

Verilog-A ADC (Part II)

Verilog-A Source (adc.va) (cont'd)

```
  analog begin
    @(cross(V(clk) - vth, +1) ) begin
      sample = V(in);
      for (i = bit - 1; i >= 0; i = i - 1) begin
        if( sample>vth ) begin
          result[i] = 5.0;
          sample = sample - vth;
        end else begin
          result[i] = 0.0;
        end
      sample = 2.0 * sample;
      end
    end
    for (i = 0; i < bit; i = i + 1 ) begin
      V(out[i]) <+ transition(result[i], dly, ttime);
    end
  end
endmodule
```