Verilog-AMS Simulation using Mentor and Cadence Tools

Prepared by

Prateek Singh Meena
05007015
Contents

Section 1
Verilog-AMS
  1.1 Verilog-AMS Introduction .......................................................... 3
  1.2 What is a Test Bench? ................................................................. 3

Section 2
Mentor Graphics Tools
  2.1 Introduction to Mentor Graphics Tools ......................................... 5
  2.2 Simulating a Verilog-AMS code using VASIM ................................. 6
    2.2.1 Example of an 8-bit ADC ...................................................... 6
      (a) Procedure ............................................................................ 7
      (b) Simulation ............................................................................ 8
      (c) Waveforms ......................................................................... 10
    2.2.2 Example of an 8-bit DAC ...................................................... 13
      (a) Procedure ............................................................................ 14
      (b) Simulation ............................................................................ 15
      (c) Waveforms ......................................................................... 17

Section 3
Cadence System Tools
  3.1 Introduction to Cadence System Tools .......................................... 19
  3.2 Simulating a Verilog-AMS code using Virtuoso AMS Designer ........ 20
    3.2.1 Example of the 8-bit ADC ...................................................... 20
      (a) Procedure ............................................................................ 21
  3.3 Simulating a Verilog-AMS code using Virtuoso Spectre ................. 21
    3.3.1 Example of the 8-bit ADC-cum-DAC ..................................... 21
      (a) Procedure ............................................................................ 24
      (b) Simulation Result ................................................................. 24
      (c) Waveform ......................................................................... 25

References .......................................................................................... 26
1.1 Verilog-AMS Introduction

Verilog-AMS is a behavioral language for analog and mixed-signal systems. Verilog-AMS is derived from IEEE std. 1364-2005 Verilog HDL. Verilog-AMS consists of the complete IEEE std. 1364-2005 Verilog HDL specification, an analog equivalent for describing analog systems (also referred to as Verilog-A), and extensions to both for specifying the full Verilog-AMS.

Verilog-AMS lets designers of analog and mixed-signal systems and integrated circuits create and use modules which encapsulate high-level behavioral descriptions as well as structural descriptions of systems and components. The behavior of each module can be described mathematically in terms of its ports and external parameters applied to the module. The structure of each component can be described in terms of interconnected sub-components. These descriptions can be used in many disciplines such as electrical, mechanical, fluid dynamics, and thermodynamics. For continuous systems, Verilog-AMS is defined to be applicable to both electrical and non-electrical systems description. It supports conservative and signal-flow descriptions by using the concepts of nets, nodes, branches, and ports as terminology for these descriptions. The solutions of analog behaviors which obey the laws of conservation fall within the generalized form of Kirchhoff’s Potential and Flow Laws (KPL and KFL). Both of these are defined in terms of the quantities (e.g., voltage and current) associated with the analog behaviors.

Verilog-AMS has the potential to shorten design cycles and increase success of more mixed signal ICs.

Some of its features are:

- Signals of both analog and digital types can be declared in the same module
- Increasing verification of analog performance at the top level of the design.
- Significantly reducing the top level simulation time.
- Creating an appropriate environment for chip architecture design.

Basic requirement for Verilog-ams code simulation, is to have a Verilog-ams code file with file extension of .vams, and a testbench also with the extension of .vams.

1.2 What is a Test bench?

A test bench is a virtual environment used to verify the correctness or soundness of a design or model. The term “testbench” usually refers to simulation code used to create a predetermined input sequence to a design, and then observe the response.

Fig. 1.1 shows how a testbench interacts with a design under verification (DUV). The testbench provides inputs to the design and watches any outputs. Notice how this is a
completely closed system: no inputs or outputs go in or out. The testbench is effectively a model of the universe as far as the design is concerned. The verification challenge is to determine what input patterns to supply to the design and what is the expected output of a properly working design when submitted to those input patterns.

Figure 1.1 – Structure of a Testbench and design under Verification

A test bench or testing workbench has four components.
1. Input: The entrance criteria or deliverables needed to perform work
2. Procedure to do: The tasks or processes that will transform the input into the output
3. Procedures to Check: The processes that determine that the output meets the standards.
4. Output: The exit criteria or deliverables produced from the workbench

Simulators available to Verilog-AMS Simulation in VLSI Lab:

- **Cadence Design Tools**
  - Cadence Affirma - Unified simulation engine for Verilog, VHDL, and SystemC.
  - Spectre - Spectre is a SPICE-class circuit simulator. Spectre comes in enhanced versions that also support RF simulation (SpectreRF) and mixed-signal simulation (AMS Designer).

- **Mentor Graphics Tools**
  - ADVance MS - ADVance MS is a language neutral, mixed-signal simulator that enables top-down design and bottom-up verification of multi-million-gate analog/mixed-signal System-on-Chip designs.

One should ask the Lab Administrator to make both the software available for him/her; otherwise you won’t be able to run them from the command line in terminal.
Chapter 2

Mentor Graphics Tools

2.1 Introduction to Mentor Graphics Tools

Mentor Graphics is a technology leader in electronic design automation (EDA), providing software and hardware design solutions. It has the broadest industry portfolio of best-in-class products and is the only EDA Company with an embedded software solution. Mentor Graphics Tools provides several different tools for different work varying from Schematic designing to simulation of mixed-signal systems.

Products Provided -

- DxDesigner a schematic editor.
- Layout Tools for printed circuit boards with programs such as PADS, Expedition and Board Station.
- ModelSim - A popular hardware simulation and debug environment
- Eldo - A high performance SPICE simulator
- ADiT (Analog Digital turbo simulator) - A fast SPICE simulator
- ADVance MS - A mixed-signal verification tool

We will be looking into Analog-Mixed signal simulation part which is done using ADVance MS. ADVance MS features a single-kernel architecture which allows you to efficiently simulate mixed designs within one consistent environment. ADMS combines four high performance simulation engines for digital, analog, mixed-signal and RF circuits in one efficient tool. This platform is built upon four high-performances, simulation technologies:-

- Eldo tool for analog large signal and frequency domain simulations
- ModelSim for digital simulations
- Mach for fast transistor-level simulations
- Eldo RF for modulated steady state simulations.

ADVance MS supports most of the design languages, including VHDL, VHDL-AMS, Verilog, Verilog-AMS, SystemC, SystemVerilog, Spice, and C, for the design and verification of mixed-signal system, and SoC. It has gained wide acceptance since its introduction five years ago. It is currently used in hundreds of customer sites.

ADVance MS also supports ADiT (Analog Digital Turbo Simulator) for fast SPICE simulation. ADiT can simulate large circuits with simulation speed 10X~ 100X faster than that of SPICE. It also allows the user to add MathWorks Simulink models to an ADVance hierarchy.
The tool ADVance MS can be invoked on the Solaris/Xilinx Platform by typing the following in the terminal:

```
vasim &
```

### 2.2 Simulating a Verilog-AMS code using ADVance MS

It requires a Verilog-AMS code file, a testbench file, a .do file and a .cmd file.

#### 2.2.1 Example of an 8-bit ADC

(One more file is included for this code which is a connect rules file):

1. **Verilog-AMS code (adc.vams)**

   ```verilog
   `include "disciplines.vams"
   `timescale 1ns / 1ps
   module adc (out, in, clk);
   parameter integer bits = 8 from [1:24]; // resolution (bits)
   parameter real fullscale = 1.0; // input range is from 0 to fullscale (V)
   parameter real td = 0; // delay from clock to output (ns)
   input in, clk;
   output out;
   voltage in;
   reg [0:bits-1] out;
   reg over;
   real sample, midpoint;
   integer i;
   always @(posedge clk) begin
     sample = V(in);
     midpoint = fullscale/2.0;
     for (i = bits - 1; i >= 0; i = i - 1) begin
       over = (sample > midpoint);
       if (over)
         sample = sample - midpoint;
       sample = 2.0*sample;
       out[i] <= #(td) over;
     end
   end
   endmodule
   ```

* ‡ Code is taken from http://www.designersguide.org/VerilogAMS/
2. **Testbench Code (testbench.vams)** *‡

```verilog
`timescale 10ns / 10ps
`include "disciplines.vams"

module testbench (in);
electrical in, gnd;
input in;
ground gnd;
reg clk;
wire [0:7] out;
integer ii;
reg [0:7] plot_out;
initial clk=0;
always #1 clk=~clk;
always @(out) for (ii=0; ii<8; ii=ii+1) plot_out[ii] <= out[7-ii];
adc adc0 (out, in, clk);
endmodule
```

3. **Defining the Connect rules (connectrules.vams)** *‡

```verilog`
`include "disciplines.vams"
connectrules my_connectrules;
    connect electrical, voltage resolveto electrical;
endconnectrule
```

4. **The .cmd file (run.cmd)** ‡

```plaintext
* .model testbench macro lang=verilogams
Y1 testbench
+ port : T1
v1 T1 0 sin (0.5 0.5 1meg 0 0)
.tran 100us 100u
```

5. **The .do file (run.do)** ‡

```plaintext
add wave -r *
run 100us
exit
```

(a) **Simulation Procedure**

Now Simulating the code ‡

- You must first create a library. Use valib to create the library in the current
directory from the Unix command line:

$ valib my_connectlib

- Define this library as the working library using the vasetlib command:
  $ vasetlib my_connectlib

- Compile the model from the Unix command line:
  $ valog -work my_connectlib connectrules.vams

- Create another Library:
  $ valib worklib

- Define this library as the working library same as done above:
  $ vasetlib worklib

- Now compile the remaining files using valog:
  $ valog adc.vams
  $ valog testbench.vams

- You can choose between using batch and interactive simulation. Use the command below for a batch simulation, use the wave viewer (EZwave) to view the results at the end of the simulation:
  $ vasin -c -cmd run.cmd -do run.do

The simulation will be shown in the following way:

(b) Simulation

# ADMS v4.5_3.1 Production Wed Mar  7 09:51:38 GMT 2007
# ELDO v6.8_3.1 Production(64 bits) Tue Mar  6 18:07:14 GMT 2007
# Software under License
# Copyright Mentor Graphics Corporation
# ---------------
# | Load starting |
# ---------------
# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/STD"."standard"
# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/IEEE"."math_real"
# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/DISCIPLINES"."physical_constants"
# Loading
"/cad/Mentor_tools/AMS_2006_2b/aol/libs/DISCIPLINES":"electromagnetic_system"

# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/IEEE","std_logic_1164"
# Loading "/home/users/prateek_sm/Desktop/listing12/worklib","testbench"
# Loading "/home/users/prateek_sm/Desktop/listing12/worklib","adc"
#
# Note: (vasim - 1309) Digital Simulation Resolution is "fs"
# "Loading worklib/MS.vloga_dig_submod_adc"
# "Loading worklib/MS.vloga_dig_submod_testbench"
# Load done
# Executing file run.do
# Start macro
#
#  ----------------------------
# | Analog Data pre-processing |
#  ----------------------------
# Analog data pre-processing done
#
#  -----------------------------
# | Summary on Analog Data |-------
# Analog kernel and ELDO devices memory space allocated (bytes): 511702
# 3 devices
# 1 nodes
# 1 input signals
# Note: Extra devices and nodes might be created during simulation (access resistors, internal nodes...).
# Note: Extra Memory might be allocated during simulation.
#  Summary done  
#
#  -----------------------------
# | Quiescent Point computation |
#  -----------------------------
# Analog DC computation starting
# Analog DC computation done
#
# Digital computation at time 0 fs starting
# Digital computation at time 0 fs done
# Digital computation at time 0 fs done
# Quiescent point done
#
#
#  -----------------------------
# | Transient analysis starting | 
#  -----------------------------
# Eldo analog kernel simulation options:
# VNTOL = 1.000000e-06
# RELTOL = 6.111111e-04
# HMIN = 1.000000e-03 ns
# HMAX = 6.666667e+01 ns
And your Simulation is completed.
This complete file tell about everything, like time taken in simulation

To see the waveform, a file with extension .wdb is created in the parent folder. This file can be opened using the EZwave. EZwave is a versatile mixed-signal waveform viewer and waveform calculation tool for display and analysis of mixed-signal results included in ADVance MS. Just open this .wdb file in EZwave, and you can all the waveforms created.

(c) Waveforms:
The waveform is on the next page:
Figure 2.1 Waveform of ADC, Sin wave is given as the input.
You can see the time when sin wave is at its lowest pt. all the bits are 0, and as it starts moving up, the first bit which goes up is the last one.
2.2.2 Example of an 8-bit DAC:-

1. Verilog-AMS code for DAC (dac.vams) *‡

```verilog
`timescale 1ns / 1ps
`include "disciplines.vams"

module dac (out, in, clk);
    parameter integer bits = 8 from [1:24]; // resolution (bits)
    parameter real fullscale = 1.0; // output range is from 0 to fullscale (V)
    parameter real td = 0.0; // delay from clock edge to output (s)
    parameter integer dir = 1 from [-1:1] exclude 0;
        // +1 triggers on rising clock edge, -1 on falling
output out;
electrical out;
input [0:bits-1] in;
input clk;
logic in, clk;
real result, aout;
integer weight;
integer i;
parameter integer idir = (dir == 1 ? 1 : 0);

always @(clk) begin
    if (clk == idir) begin
        aout=0.0;
        weight = 2;
        for (i=bits-1; i>=0; i=i-1) begin
            if (in[i]) aout = aout + fullscale / weight;
            weight = weight * 2;
        end
        result = #(td / 1.0E-9) aout;
    end
end
analog begin
    V(out) <+ result;
end
endmodule
```

*‡ Code is taken from http://www.designersguide.org/VerilogAMS/
2. Testbench code (testbench.vams) 

```
`timescale 10ps / 1ps
module testbench ();
    reg clk;
    reg [0:7] in;
    integer i,ii;
    electrical out;
        initial begin
            clk=0;
            in=0;
            i=0;
        end
    always #100 clk=~clk;
    always begin
        #333
            wait (!clk);
            in=i;
            i=i+1;
            if (i == 256) i=0;
            for (ii=0; ii<8; ii=ii+1) in[ii] <= in[7-ii];
        end
    dac dac0 (out, in, clk);
endmodule
```

3. The Command file (run.cmd) 

```
.*
.tran 1us 5us
```

4. The .do file (run.do) 

```
add wave -r *
run -all
exit
```

(a) Procedure:
Now Simulating the code, it’s done the same way this time also 

- Create another Library:
  
  $ valib worklib

- Define this library as the working library same as done above:
  
  $ vasetlib worklib

* ‡ Code is taken from http://www.designersguide.org/VerilogAMS/
• Now compile the remaining files using valog:
  
  $ valog dac.vams

  $ valog testbench.vams

• You can choose between using batch and interactive simulation. Use the command below for a batch simulation

  $ vasim -c -cmd run.cmd -do run.do

(b) Simulation:

The simulation will be shown in the following way:

# ADMS v4.5_3.1 Production Wed Mar  7 09:51:38 GMT 2007
# ELDO v6.8_3.1 Production(64 bits) Tue Mar  6 18:07:14 GMT 2007
# Software under License
# Copyright Mentor Graphics Corporation
#
# ---------------
# | Load starting |
# ---------------
# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/STD"."standard"
# Loading "/cad/Mentor_tools/AMS_2006_2b/aol/libs/IEEE"."std_logic_1164"
# Loading "/home/users/prateek_sm/Desktop/listing09/worklib"."testbench"
# Loading "/home/users/prateek_sm/Desktop/listing09/worklib"."dac"
# Note: (vasim - 1309) Digital Simulation Resolution is "fs"
# "Loading worklib/MS.vloga_dig_submod_dac"
# "Loading worklib/MS.vloga_dig_submod_testbench"
# Load done
# Executing file run.do
# Start macro
#
# Analoga Data pre-processing done
#
# Summary on Analog Data |-------------------------------------------
# Analog kernel and ELDO devices memory space allocated (bytes): 501906

# 2 devices
# 1 nodes
# 0 input signals
# Note: Extra devices and nodes might be created during simulation (access resistors, internal nodes...).
# Note: Extra Memory might be allocated during simulation.
#  "Summary done"  
#  "-----------------------------  
#  "Quiescent Point computation"  
#  "-----------------------------  
#  "Analog DC computation starting  
#  "Analog DC computation done  
#  "Digital computation at time 0 fs starting  
#  "Digital computation at time 0 fs done  
#  "Digital computation at time 0 fs done  
#  "Quiescent point done  
#  "-----------------------------  
#  "Transient analysis starting"  
#  "-----------------------------  
# Eldo analog kernel simulation options:  
#  "VNTOL = 1.000000e-06  
#  "RELTOL = 6.111111e-04  
#  "HMIN = 1.000000e-03 ns  
#  "HMAX will be set later during simulation and may vary  
#  "EPS = 5.000000e-04  
#  "Starting analog time is 0.000000 ns  
#  "Simulation time for "run -all" is 5.000000E+03 ns  
#  "5,000,000,000 fs 100.00%  
#  "End macro  
#  "Transient analysis done  
#  "Global Cpu Time 0h 0mn 0s 320ms  
#  "Global Elapsed Time 0h 0mn 26s // Total time taken for simulation  
#  "-----------------------------| Simulation Information |-----------------------------  
#  "Analog kernel and ELDO devices memory size allocated in bytes 1069185  
#  "2 devices  
#  "1 nodes  
#  "0 MOS or BIP model calls  
#  "2703 steps computed  
#  "Simulation Information done  
#  "Simulation finished at time 5 us  
The file created with the extension of .wdb (run.wdb) can be viewed using EZwave.
(b) Waveforms:

The simulation results are:

Figure 2.3 Waveform of a DAC giving Saw tooth wave as the output
Figure 2.4 Waveform within a span of 35ns
3.1 Introduction to Cadence Tools, Virtuoso

Cadence Design Systems is the world's leading EDA company. For years it had been the largest company in the EDA industry. Cadence provides a variety of products targeted at various types of design and verification tasks which include:

- **Virtuoso Platform**: Tool for designing full custom integrated circuits, used for analog, mixed-signal, RF, and standard cell designs.
- **Encounter Platform**: Tools for creation of digital integrated circuits. This includes floor planning, synthesis, test, and place and route.
- **Incisive Platform**: Tools for simulation and functional verification of RTL including Verilog, VHDL and SystemC based models. Includes formal verification, formal equivalence checking, hardware acceleration, and emulation.
- **Allegro Platform**: Tools for co-design of integrated circuits, packages, and PCBs.
- **OrCAD/PSpice**: Tools for smaller design teams and individual PCB designers.

In Cadence, Virtuoso Multimode Simulation is a comprehensive design and verification solution that combines the industry's leading SPICE, FastSPICE, RF, and mixed-signal simulators in a unique shared licensing scheme. It meets the changing simulation needs of designers as they move through the design cycle, from architecture exploration to block-level development to RF design and to final full-chip verification. Virtuoso Multi-Mode Simulation contains:

- **Cadence Virtuoso Spectre Circuit Simulator**: It which provides fast, accurate SPICE-level simulation for tough analog, RF and mixed-signal circuits.
- **Cadence Virtuoso UltraSim Full-Chip Simulator**: It is the Cadence FastSPICE circuit simulator that provides speed, capacity, and accuracy when verifying large custom, analog mixed-signal, RF, memory, and SoC designs.
- **Cadence Virtuoso AMS Designer Simulator**: It is a mixed-signal simulation solution for the design and verification of the largest and most complex analog, RF, memory and mixed signal SoCs. It is also integrated into the Incisive functional verification platform for mixed signal verification within the digital verification environment.
3.2 Simulating a Verilog-AMS code using Virtuoso AMS Designer Simulator

It requires a Verilog-AMS code file, a testbench file, a .tcl file and a .scs file. The .scs file is similar to the .cmd file, and .tcl file is similar to the .do file as we saw in ADVance MS.

The tool Virtuoso AMS Designer Simulator can be invoked on the Solaris/Xilinx Platform by typing the following in the terminal:-

```
ncsim &
```

We do not have NCSIM currently in our VLSI lab, which is why we were unable to do the simulation using Virtuoso AMS Designer Simulator, but explaining the complete method of simulating a Verilog-AMS code.

3.2.1 Example of an 8-bit ADC (One more file is included for this code which is a connect rules file):

Two other setup files, listed below, are required before you can start using the tools. Copy the contents from here into your design folder. These files specify the path to libraries and also create a user work library. In this case the user work library has been called worklib but can be given any name.

1. **hd1.var**
   `SOFTINCLUDE $CDS_INST_DIR/tools/inca/files/hdl.var`
   `DEFINE USE_NEW_SIMWAVE_WINDOW ON`
   `DEFINE EDITOR vi`
   `DEFINE CDS_TEXT_EDITOR vi`
   `DEFINE WORK worklib`

2. **cds.lib**
   `SOFTINCLUDE $CDS_INST_DIR/tools/inca/files/cds.lib`
   `DEFINE worklib ./worklib`

---

1. **Verilog-AMS code (adc.vams) **
   This code is the same one used in the Simulation using Mentor Tools.

2. **Testbench code (testbench.vams)**
   This code is slightly different from the last one.

`timescale 10ns / 10ps`
`include "disciplines.vams"`
module testbench ();
    electrical gnd;
    ground gnd;
    reg clk;
    wire [0:7] out;
    integer ii;
    reg [0:7] plot_out;
    initial clk=0;
    always #1 clk=~clk;
    always @(out) for (ii=0; ii<8; ii=ii+1) plot_out[ii] <= out[7-ii];
    adc adc0 (out, in, clk);
    vsource #(type("sine"), .ampl(0.5), .dc(0.5), .freq(1M)) v0 (in, gnd);
endmodule

3. The .scs file (run.scs) *
   *
   transient tran stop=100us

4. The .tcl file (run.tcl) *
   database -open waves -into waves.shm -default
   probe -create testbench -depth all -shm -waveform
   run 100us
   exit

(a) Simulation Procedure:

- Same as we did in ADVance MS, first create the library
  $ mkdir worklib
  $ mkdir my_connectlib

- compiling the connect rules
  $ ncvlog -ams -work my_connectlib connectrules.vams
  $ ncvlog -ams adc.vams
  $ ncvlog -ams testbench.vams
  $ ncelab testbench my_connectrules -timescale 10ps/1ps

- Use the Command below for batch simulation
  $ ncsim testbench -messages -analogcontrol run.scs -input run.tcl

* † Code is taken from http://www.designersguide.org/VerilogAMS/
A folder with extension .raw is created in the parent folder containing all the data regarding the simulation, at the waveform can be viewed using Wavescan, a waveform viewer in Cadence.

The 2\textsuperscript{nd} Example can also be simulated in the similar way.

### 3.3 Simulating a Verilog-AMS code using Virtuoso Spectre Circuit Simulator

It requires a Verilog-AMS code file and a testbench.scs file this .scs file is the circuit cum testbench file.

#### 3.3.1 Example of an 8-bit ADC cum DAC:

This is performing both of the operations, $V(\text{in})$ is given as input and ADC convert it into 8-bits, those 8-bit acts as a input to the DAC, and DAC gives $V(\text{out})$ as the output.

1. **Code of ADC (adc.vams) *‡**

   ```verbatim
   `include "disciplines.vams"
   module adc(out, in, clk);
   `define bits 8 // resolution (bits)
   parameter real fullscale = 1.0; // input range is from 0 to fullscale (V)
   parameter real td = 0; // delay from clock edge to output (s)
   parameter real tt = 0; // transition time of output (s)
   parameter real vdd = 5.0; // voltage level of logic 1 (V)
   parameter real thresh = vdd/2; // logic threshold level (V)
   parameter integer dir = 1 from [-1:1] exclude 0; // 1 for rising edges, -1 for falling
   input in, clk;
   output [0:`bits-1] out;
   voltage in, clk;
   voltage [0:`bits-1] out;
   real sample, midpoint;
   integer result[0:`bits-1];
   integer i;
   analog begin
   @(cross(V(clk)-thresh, dir) or initial_step) begin
   sample = V(in);
   midpoint = fullscale/2.0;
   for (i = `bits - 1; i >= 0; i = i - 1) begin
   if (sample > midpoint) begin
   end
   end
   end
   ```

*‡ Code is taken from [http://www.designersguide.org/VerilogAMS/](http://www.designersguide.org/VerilogAMS/)
result[i] = 1;
sample = sample - midpoint;
end else begin
    result[i] = 0;
end
sample = 2.0*sample;
end
end

generate i (`bits-1,0) begin
    V(out[i]) <+ transition(result[i] ? vdd : 0.0, td, tt);
end
derandom
endmodule
`undef bits

2. Code of DAC (dac.vams) *

`include "disciplines.vams"

module dac(out, in, clk);
`define bits 8 // resolution (bits)
parameter real fullscale = 1.0;// output range is from 0 to fullscale (V)
parameter real td = 0; // delay from clock edge to output (s)
parameter real tt = 0; // transition time of output (s)
parameter real vdd = 5.0; // voltage level of logic 1 (V)
parameter real thresh = vdd/2; // logic threshold level (V)
parameter integer dir = 1 from [-1:1] exclude 0;
    // 1 for rising edges, -1 for falling
output out;
input [0:`bits-1] in;
input clk;
voltage out, clk;
voltage [0:`bits-1] in;
real aout;
integer weight;
integer i;

analog begin
    @(cross(V(clk) - thresh, dir) or initial_step) begin
        aout = 0;
        weight = 2;
        generate i (`bits - 1, 0) begin
            result[i] = 1;
            sample = sample - midpoint;
            end else begin
                result[i] = 0;
            end
            sample = 2.0*sample;
        end
    end
    generate i (`bits-1,0) begin
        V(out[i]) <+ transition(result[i] ? vdd : 0.0, td, tt);
    end
end
endmodule
`undef bits

* Code is taken from http://www.designersguide.org/VerilogAMS/
```verilog
if (V(in[i]) > thresh) begin
    aout = aout + fullscale/weight;
end
weight = weight*2;
end
endmodule

V(out) <+ transition(aout, td, tt);
end
endmodule

`undef bits

3. Code of the Testbench (testbench.scs) *‡

```verilog
// Test adc model
simulator lang=spectre

ahdl_include "adc2.vams"
ahdl_include "dac2.vams"

Vin  (in 0) vsource type=sine ampl=0.5 dc=0.5 freq=1
Vclk (clk 0) vsource type=pulse val0=0 val1=1 period=1ms
ADC (b7 b6 b5 b4 b3 b2 b1 b0 in clk) adc vdd=1 td=100us tt=100us
DAC (out b7 b6 b5 b4 b3 b2 b1 b0 clk) dac vdd=1 td=100us tt=100us
sineResp tran stop=1
```

(a) Procedure of Simulating Using Spectre

spectre testbench.scs

(b) Simulation Result:

spectre (ver. 5.10.41.031208 -- 12 Mar 2008).
Includes RSA BSAFE(R) Cryptographic or Security Protocol Software from RSA Security, Inc.
Simulating `testbench.scs' on vlsi27 at 12:02:09 PM, Thur Apr 23, 2009.
Compiling ahdlcmi module library.
Failed to compile ahdlcmi module library, see adc2.vams.ahdlcmi/ for details
Could not open ahdlcmi module library
	adc2.vams.ahdlcmi/obj/Linux2.6.9-22.ELsmp+gcc/optimize/libahdlcmi.so
	adc2.vams.ahdlcmi/obj/Linux2.6.9-22.ELsmp+gcc/optimize/libahdlcmi.so:
cannot open shared object file: No such file or directory

*‡ Code is taken from http://www.designersguide.org/VerilogAMS/
Compiling ahdlcmi module library.
Failed to compile ahdlcmi module library, see dac2.vams.ahdlcmi/ for details
Could not open ahdlcmi module library

dac2.vams.ahdlcmi/obj/Linux2.6.9-22.ELsmp+gcc/optimize/libahdlcmi.so
dac2.vams.ahdlcmi/obj/Linux2.6.9-22.ELsmp+gcc/optimize/libahdlcmi.so:
cannot open shared object file: No such file or directory

Notice from spectre during topology check.
Only one connection to node `out'.

Circuit inventory:
- nodes 11
- equations 22
- ahd1 simulator 1
  - ade 1
  - dac 1
- vsource 2

**************************************************
Transient Analysis `sineResp': time = (0 s -> 1 s)
**************************************************

Important parameter values:
- start = 0 s
- outputstart = 0 s
- stop = 1 s
- step = 1 ms
- maxstep = 20 ms
- ic = all
- skipdc = no
- reltol = 1e-03
- abstol(I) = 1 pA
- abstol(V) = 1 uV
- temp = 27 C
- tnom = 27 C
- tempeffects = all
- errpreset = moderate
- method = traponly
- lteratio = 3.5
- relref = sigglobal
- cmin = 0 F
- gmin = 1 pS
- maxrsd = 0 Ohm
- mos_method = s
- mos_vres = 50 mV
Number of accepted tran steps = 15580.
Initial condition solution time = 0 s.
Intrinsic tran analysis time = 920 ms.
Total time required for tran analysis `sineResp' was 920 ms.
Aggregate audit (12:02:23 PM, Thur Apr 23, 2009):
Time used: CPU = 1.07 s, elapsed = 14 s, util. = 7.63%.
Virtual memory used = 4.06 Mbytes.
spectre completes with 0 errors, 0 warnings, and 1 notice.

(c) Waveform:
A .raw file is created in the parent folder, which can be opened using Wavescan and waveform can be viewed
   The Red line is the V(out)
   Just above it is V(in), and the rest are the bits.
References:

- ADVance User's Manual
- ADVance MS Datasheet
- Virtuoso AMS Designer Datasheet
- http://www.designers-guide.org/VerilogAMS/