



# **Design and Verification of Nanometer SoCs using AMS Designer**

**Tony Blake**

**Design Engineer, Silicon & Software Systems ([www.s3group.com](http://www.s3group.com))**

**12 September 2005**



# Contents

- Introduction
- Background – Tool & Language
- Design Flow
- Common Testbench Infrastructure
- Block Analog Modelling
- Chip/System Level
- Results
- Conclusions



# Introduction – Motivation for using Verilog-AMS

- SoC(System on Chip) contains:
  - RF and Analog, processor core, memories and peripherals
  - A number of control loops between analog and digital
  - Datapaths and data processing across analog and digital
- Benefits of using Verilog-AMS:
  - Simulation flow for rapid design convergence
  - Common Testbench Infrastructure – add on to standard digital one
  - Analog interface and connectivity of analog modules derived directly from schematics(block functionality modelled)
  - Comprehensive system-level simulations and simulation of digital/analog interface with self-checking e.g. Ultrasim simulation of one analog sub-system took 3 weeks – Verilog-AMS took 3 hours.



## Background – What is Virtuoso AMS Designer ?

- Top-down system-on-chip simulation for complex mixed-signal, mixed-language designs
- Two components: AMS Designer Environment & Simulator
- Command-line or GUI interfaces
- A **single executable simulator** - NC-Sim and Spectre
- INCA(Interleaved Native Compiled Architecture)
- Mixed-language debugger & mixed-signal waveform display



## Background – Verilog-AMS Language

- New mixed-domain language based on Verilog HDL and Verilog-A
- Discrete & continuous signals can be simulated and displayed together
- Data access & event control from both domains
- Variables can only be assigned in one context
- Disciplines – electrical & logic

# Example of Verilog-AMS code

```

`include "disciplines.vams"
module pga ( code, en, outm, outp, inm, inp, cmode);
  output outp, outm; input inp, inm, cmode;
  input [1:0] code; input en;
  electrical outm, outp, inm, inp, cmode;
  logic [1:0] code; logic en;

  real gain [0:3]; real gain_val, in_diff, out_diff, v_out;
  integer gain_setting;
  parameter real p_rise_fall = 1u from [0:10u];

  initial begin
    gain[0] = 1; gain[1] = 0.75; gain[2] = 0.5; gain[3] = 0.25;
  end

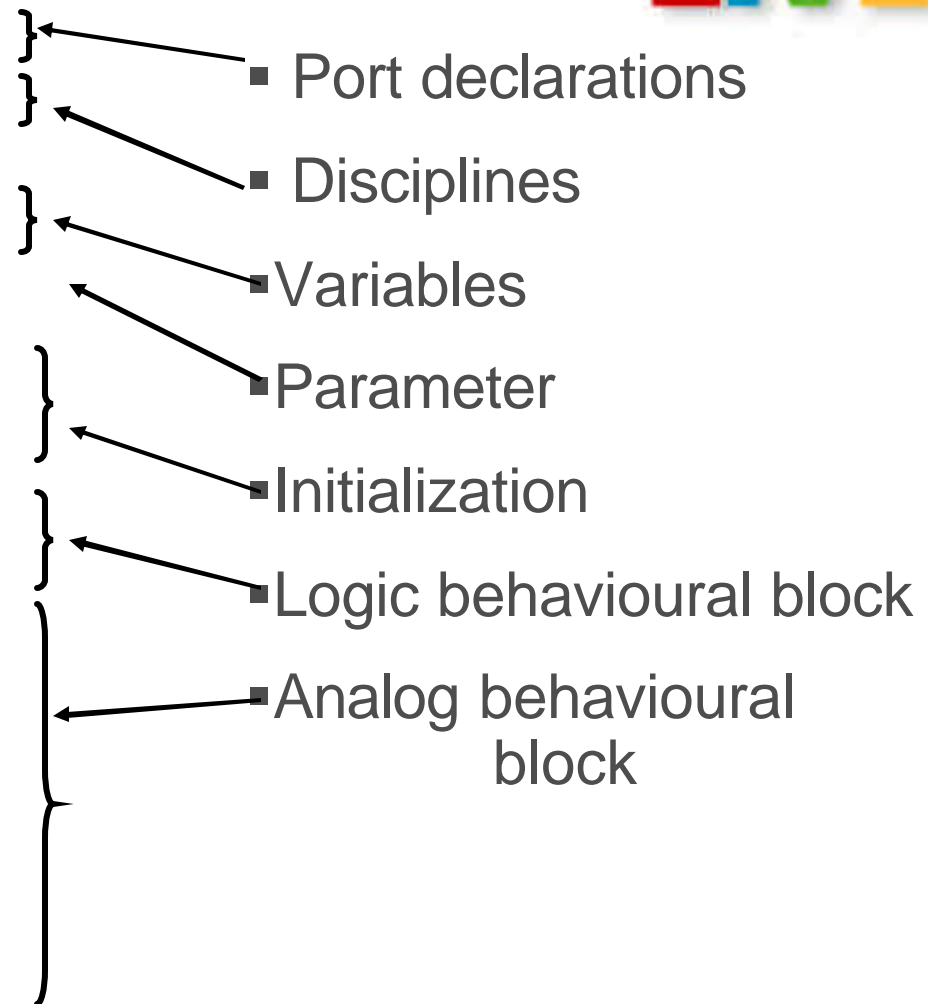
  always @ (code) begin
    gain_setting = code[1:0]; gain_val = gain[gain_setting];
  end

  analog begin
    in_diff = V(inp, inm);
    if (en) out_diff = -(gain_val) * in_diff;
    else out_diff = 0;

    v_out = transition(out_diff, 0, p_rise_fall);
    V(outp) <+ V(cmode) - v_out/2;
    V(outm) <+ V(cmode) + v_out/2;
  end
endmodule

```

## Programmable Gain Amplifier

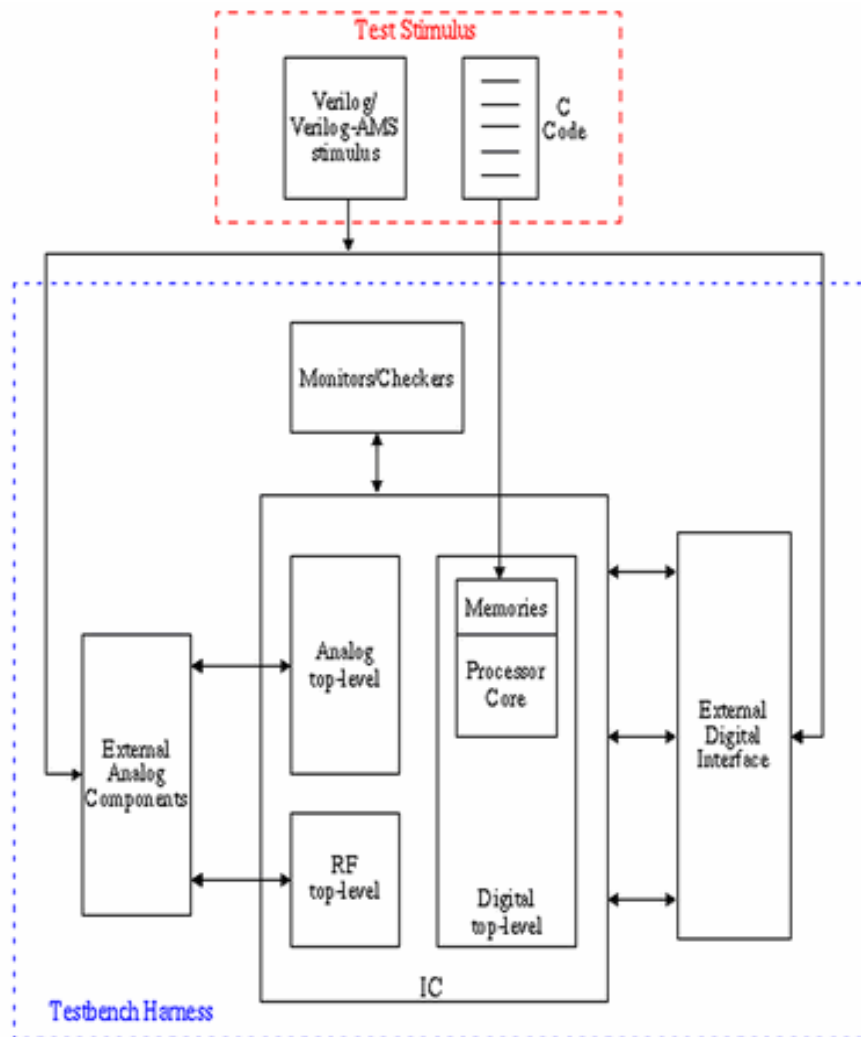




## Design Flow

- Develop Verilog-AMS behavioural views for analog & RF blocks
- Calibrate against final schematic views
- Combine behavioural views with structural netlists for higher level blocks
- Link all Verilog-AMS views into common testbench structure i.e. the digital verification environment.

# Common TestBench Infrastructure



## Test Stimulus

- Processor code (C and/or Assembly) to stimulate the design & provide self-checking capability
- Verilog-AMS file to control the external interface driver/monitors, analog components and define parameters for the test e.g. clk freq, amp gain levels etc.

## Testbench Harness

- Combination of Verilog & Verilog-AMS
- Reused for all sub-system and system level testing
- Instantiation of IC under test
- Driver/monitors for external digital interface
- External analog components
- Verilog-AMS behavioural models for analog/RF
- Connect modules – inserted automatically by the simulator to handle transitions at every interface between domains
- Testbench monitors/checkers



## Common Testbench Infrastructure ....

- Customer use cases for the system can be verified across the different domains.
- A compiler directive can be used to switch in the Verilog-AMS views and external analog components thus removing their speed overhead for digital only simulations e.g.

```
`ifdef AMS  
    vsource #(.type("dc"), .dc(1.2)) v0  
        ( sup_1p2, agnd );  
`endif
```



# Block Analog Modelling

- Keep in mind - Ultimate use is in chip level simulations

## **Guidelines:**

- *Use Top-down design:* get libraries, block names & port lists defined at the start.
- *Model at the appropriate level:* not too low but high enough that structural Verilog-AMS netlisting can be used for the higher level blocks



## Block Analog Modelling ...

- *Creating Verilog-AMS behavioural views:* generate *verilogams* view directly from the symbol - ensures correct pin names & port order. Use the on-line editor for code syntax checking.
- *Calibration:* Do block level testbench work within Cadence - allows same testbench as for schematic i.e. Verilog-AMS view is verified with circuit level simulator. Keep external analog components technology independent.
- *Complexity:* Model all core functionality & test modes. Finer details of schematic need not be captured.



## Block Analog Modelling ...

- *Embedded self-checking capability*: bias currents and power supply levels can be verified locally e.g. the signal *curr\_ok* can be used to control output assignments later in the code.

```
always #50 begin
    i_src = I(i_bias);
    if ((i_src < (0.99 * 1u)) || (i_src > (1.01 * 1u)))
        curr_ok = 1'b0;
    else
        curr_ok = 1'b1;
end
```



## Block Analog Modelling ...

- *Chip convergence*: Digital control signals could be x or z at start up – accessing an x or z in an analog process is an error! Use the equality “===” or inequality “!==" operators to overcome e.g.

```
if (enable === 1)
    V(out) <+ V(ref);
else
    V(out) <+ 0;
```



## Block Analog Modelling ...

- *Connect Modules*: minimize their use – simulation overhead for high speed signals.
  - Discipline resolution issues can be difficult to resolve.
  - Tcl command “scope –aicms –all –recu”.
  - Implement the A2D or D2A locally in model.
  - Always understand the interface involved.
- *Design for change*: Avoid hard-coding values - use variables and parameters. *Defparam* statement to vary testcase operating conditions.



# Block Analog Modelling ...

## Guidelines for simulation speed

- *Model in the discrete domain*: as much as possible - key point. Must avoid a major simulation slow-down when the Verilog-AMS is included in the chip design.
- *Limit use of the "cross" function* – major user of cpu time.
- *NC Profiler*: use its report to see what is slowing down the simulation e.g. VCO o/p signals are analog but a significant speed increase can be achieved by treating them as logic - the information of frequency & phase is maintained and modelling of the following blocks is simpler e.g. buffers and dividers.

# Block Analog Modelling ...



## Guidelines for simulation speed ...

- *Switching activity*: minimize high speed switching activity e.g. use the block enable to gate the clock for the *always* statement.
- *Simulator settings*: e.g. *Errpreset* has a range of settings which impact the simulation speed & accuracy – investigate once the efficient block level modelling has already been done!

# Block Analog Modelling ...

## Guidelines for simulation speed ...

- *Time constants*: Rise/fall times on reference nodes driving many circuits should be updated less frequently e.g. decoupling the bandgap voltage below only impacts each connected circuit every 100us instead of at every timepoint during the rise/fall transitions.

```

real vs_ref, vt_ref, vp_ref;
parameter real t_rise_fall = 1.5m from [0:inf];
parameter real t_speriod = 100u from [0:inf];
parameter real t_del = 500p from [0:inf];
analog begin
  vs_ref = V(BGAP);
  vt_ref = transition(vs_ref,t_del, t_rise_fall, t_rise_fall);
  @(timer(0,t_speriod)) vp_ref = vt_ref;
  V(VREF) <+ vp_ref;
end

```

→ Can also adjust the parameters at testbench level to improve simulation speed.



## Chip/System Level

- *Technology components*: Structural AMS netlists for the higher level blocks may contain resistors, decoupling caps, antenna diodes etc. Create Verilog-AMS models for these.
- *Tie-off cells*: Schematic designers should use these for hardwiring signals *hi* or *lo*. Create Verilog-AMS models for them – limits use of connect modules.
- *Pad cells*: Create Verilog-AMS models for these – limits use of connect modules and technology components. Use embedded power supply voltage level checking.
- *Switching activity*: enable high activity monitors only when needed e.g. use a gated strobe in the testcase.
- *Timescale*: Have only one assignment of ``timescale`.



## Chip/System Level ...

- *Convergence*: The verilog “*force*” & “*release*” operators can be used on the problem digital control lines to put the chip into an initial known state until the reset state is established.

```
initial begin
    force `TOP_CELL.enable = 0;
    #40;
    release `TOP_CELL.enable;
end
```

→ The “*force*” command can also be used to shutdown switching activity in a part of the chip later in the simulation.



# Analog Monitors

**Chip level regression runs** - eliminate visible checking of the analog waveforms to ensure quick verification turn-around and accuracy/repeatability of the checks.

- *Probing analog signals:*

- At testbench level can probe directly if signal declared as electrical.
- Lower in the hierarchy the signals must first be sampled at block level in the discrete domain -> can then pick up their value at testbench level using a hierarchical reference e.g.



# Analog Monitors

```
real v_p, v_n;  
always @ (negedge clk) begin  
    v_p = V(afe_p);  
    v_n = V(afe_n);  
end
```

Sampling signals at block level e.g. A/D inputs

```
real v_rx_p, v_rx_n;  
always @(posedge strobe) begin  
    v_rx_p = `RX_TOP.I1.I2.I6.I22.v_p;  
    v_rx_n = `RX_TOP.I1.I2.I6.I22.v_n;  
  
    $fdisplay(filenb3, "%5.3f  %5.3f ", v_rx_p, v_rx_n);  
end
```

Reading and recording signals at testbench level



## Analog Monitors ...

- *Self-checking:*

- Testbench should have code to check the simulation results giving a Pass or Fail message.
- Analog signals evaluated within a tolerance range.
- Log files with controlled strobing - another option. Compare results vs set of reference ones.
- Partition the results into a number of files e.g. interface control signals, DC signals, testmodes, system1/2 etc.
  - > can see what is happening across the chip at a given moment.

# Analog Monitors ...

- *High speed signals*: Checking of frequency & phase.
  - Use high speed source signal with controlled strobing e.g. VCO clock or one created in the testbench e.g. in the following log file the differential signals PP, PPx track with KK\_I, KK\_Ix and are in quadrature with KK\_Q, KK\_Qx.

strobe	PP,x	KK_I,x	KK_Q,x	clk_freq(GHz)
1	xx	xx	xx	0
2	10	10	01	1.5723
3	10	10	10	1.5723
4	01	01	10	1.5723
5	01	01	01	1.5723
6	10	10	01	1.5723
7	10	10	10	1.5723
8	01	01	10	1.5723
9	01	01	01	1.5723



## Results

- *Simulation speed:* Regression simulations run in parallel on a Linux server farm - results in a couple of hours for testcases covering tens of milli-seconds.
- *Significant bugs found:*
  - One mixed signal system found to have loop instability.
  - Loading of a power ramp from the digital had an incorrect final value which caused the ramp to collapse.
  - Architectural flaw in chip voltage/current biasing network.
  - Polarity of one diff pair signal got inverted high up in the hierarchy.



## Results

- *Initialization scenarios*: useful for checking these since the impact across the SoC could be immediately seen e.g. impact of analog time constants on customer use cases.



## Conclusions

- *AMS Designer* allows a common testbench infrastructure with self-checking for the analog, RF and digital.
- Vastly improved communication between the design groups.
- Incorporating the analog & RF views into the standard digital simulation flow is an easy step.
- Analysis of complex RF/analog/digital loops is possible – an essential capability when targeting first time right silicon in advanced communication SoC design.

Questions ?

