Assertion-based Verification for Analog and Mixed Signal Designs

Srinivas Aluri
Texas Instruments
Dallas TX, USA
s-aluri@ti.com

Abstract: Assertion-based verification approaches have been making great strides into mixed signal verification environments. As the design complexity increases so does the means to verify it. Verification engineers have been using assertion-based approach in the analog domain as well. Most of the effort in this domain has been to code assertions on analog models which are abstract representation of a real circuit. This paper shows a methodology of coding assertions directly on electrical ports of a schematic sub circuit and hence can be bound to a sub circuit, thereby greatly increasing modularization and facilitating re-use.

Introduction: We are very familiar with the concept of coding checkers without touching the design by binding the code to it using verification units in the digital domain. This methodology is gaining traction in the analog and mixed signal domains. By using vunit, verification engineers can bind assertions to abstract models of analog or interactions between digital and models of analog. There are scenarios where an analog block is not modelled and is run in schematic form and verification checks needs to be performed in a complete analog domain. We should be able to apply the same techniques to effectively verify the design without slowing down the simulation speed. The work shown in this paper specifically targets such scenarios and shows how assertion techniques can be used for verifying analog parameters.

Binding Assertions: Fig.1 shows the block diagram of mixed signal DUT. Assertions can be bound as below for the shown mixed signal design.

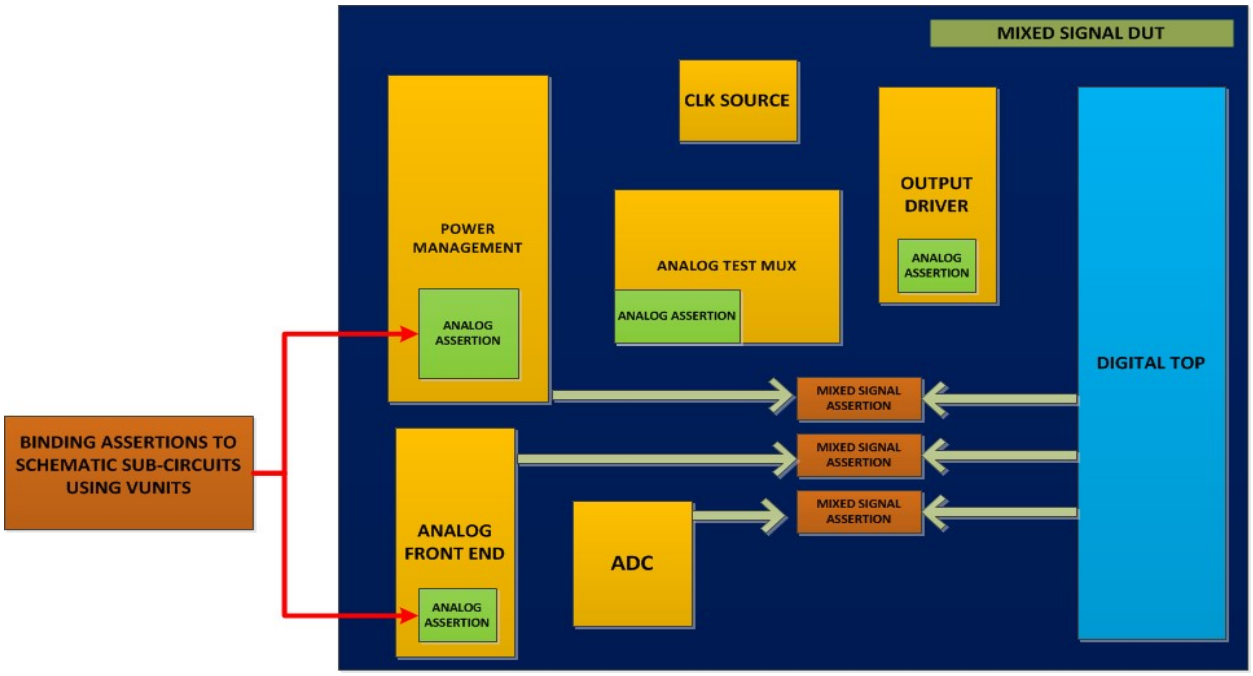

Fig 1: Block diagram showing Assertions bound to sub-circuits in Mixed signal DUT

a. *Bind assertions to digital modules and instances*
b. *Bind assertions to behavioral modules of analog blocks*
c. *Bind assertions to top level so that you can write mixed signal assertions by accessing signals from both analog and digital domain*
d. *Bind assertions directly to analog module (not model).*

The code below shows module name based assertion binding on analog module mentioned in (*d*) above. Vunit and relevant declarations of electrical ports on which checking needs to be performed are shown.

Example Code- I:

The snippets of code below shows some important steps followed while using this methodology

a. Ports of the sub-circuit on which assertions are to be coded using vunit needs to be explicity declared as electrical as shown in fig.2.
b. Assertions can be written on hierarchial nets as well, but you need to use the name of the port which the net is connected to write an assertion, and that port needs to be declared as electrical.

```
// Vunit declaration with module name based binding to PM_TOP
vunit pm_top_vunit(PM_TOP) {

/// declaring ports of PM_TOP explicitly of type electrical on which assertions are coded

electrical REF_1p2,  REF_2p5, REF_2p5_DAC, REF1p2_HV, HV_REF_0p6, DVDD_OUT;

// internal nets can also be included in assertions but they need to be ports of internal hierarchical cells

electrical IREF_GEN.VREF_2p5V, IREG_DVDD.EN, IVREF_IB_GEN.SD, IREF_GEN.IPTAT_1U, IVREF_IB_GEN.IPTAT_1p6u;
```

Fig 2: Code showing Vunit declaration and explicit port declaration on which assertions are planned.

c. Before writing assertions on the analog domain signal, we need to convert them to digital, for speed and tool limitations. Fig.3 shows the required conversion on voltage and current signals.

```
electrical REF_1p2,  REF_2p5, REF_2p5_DAC, REF1p2_HV, HV_REF_0p6, DVDD_OUT;

// Checking if Analog signal is in expected range and converting it into a digital signal
// Converting REF_1p2 → ref_1p2_in_range which can be used in assertion.
`V_IN_RANGE(REF_1p2,high_thr_limit,low_thr_limit,`vdelta,`ttol,`vtol,enable_v_assert_trigger,ref_1p2_in_range)

// Checking if Analog signal is in expected range and converting it into a digital signal
// Converting curr_iztc_500n  ->  curr_iztc_500n_in_range  which can be used in assertion.
`I_IN_RANGE(`curr_iztc_500n,high_thr_limit,low_thr_limit,`idelta,`ittol,`itol,enable_i_assert_trigger,curr_iztc_500n_in_range)
```

Fig 3: Code showing Vunit declaration and explicit port declaration on which assertions are planned.

REF_1p2 which is an electrical signal is monitored and converted to an event driven signal by using a custom macro shown in fig.4. By using appropriate threshold and tolerances in voltage and time analog variations are converted to digital signal events, which is ref_1p2_in_range in this case. In the case of current, we cannot directly use the port name as in the case of voltage, but instead need to use $cgav system task. Below is the example

```
`define curr_iztc_500n      `get_abs($cgav("<Hierarchial_DUT_path>.IZTC_500n","flow"))
```

Below is the detailed code of the macros used in the above code.

```
`define V_IN_RANGE(Node,hthr,lthr,vdelta,ttol,vtol,enable,digout) \
always @(absdelta(V(Node),vdelta,ttol,vtol,enable)) begin \
if((V(Node) > hthr) || (V(Node) < lthr)) digout = 0; \
else digout = 1; \
end
```

```
`define I_IN_RANGE(exp,hthr,lthr,edelta,ttol,etol,enable,digout) \
always @(absdelta(exp,edelta,ttol,etol,enable)) begin \
if((exp > hthr) || (exp < lthr)) digout = 0; \
else digout = 1; \
```

Fig 4: Customized macro code used in converting continuos signals to discrete signals.

After converting the analog signals to digital domain, we can use those signals and code psl assertions completely in the digital domain. Fig.5 shows an example of voltage and current checking assertions. The voltage assertion in example-1 is checking for ref_1p2 voltage in range, if vddp5v is in expected range. This assertion is triggered each time there is a change in ref_p12. In example-2 the assertion is checking for the dvdd_mon signal in the expected range, if ref1p2_hv, vdd5v_por and dvdd are in range. This assertion is triggered whenever an event is created by the absdelta function. In example-3 iptat_1u current is checked to see if it is in the expected range when vddp5v is in range. This assertion is triggered when there is a change in iptat current or if there is a change in vddp5v.

```
// Voltage Assertions

Example – 1
// CHECKER: when VDDP5V in expected range ===> Check for REF_1p2 in expected range
// psl pm_top_ref_1p2_assert: assert always (((vddp5v_in_range) -> (ref_1p2_in_range))) @(ref_1p2_in_range);

Example -2 Where absdelta is used to create events to trigger assertions

// CHECKER: when REF1p2_HV && VDD5V_POR && DVDD are in expected range ===> -0.01<DVDD_MON_OV<0.01
//  psl pm_top_dvdd_mon_ov_assert: assert always (((ref1p2_hv_in_range && vddp5v_por_in_range && dvdd_in_range) ->
((V(DVDD_MON_OV) > -0.01) && (V(DVDD_MON_OV) < 0.01)))) @(absdelta(V(DVDD_OUT),`vdelta,`ttol,`vtol,enable_v_assert_trigger));

// Current Assertions

Example -3 Assertion showing expected bias current check
// Checker: when VDDP5V is in range ===> check current on IPTAT_1U is at 1uA
//  psl pm_top_refgen_1p2_iptat_1u_curr_assert: assert always ((vddp5v_in_range) -> (curr_refgen_1p2_iptat_1u_in_range))
@(curr_refgen_1p2_iptat_1u_flag or vddp5v_in_range_dly);
```

Fig 5: Example code showing assertions on analog parameters.

Example code – II: Input and Output Analog testmux is a very standard feature. Oftentimes these testmuxes have large number of inputs to select from and we can effectively verify the functionality using the assertion technique discussed above.

a. For example lets say if a regulated voltage needs to be observed on testmux we can write an assertion shown in fig.6 to connect the source which in this case is the output of an LDO to output of the output analog testmux, given that the mux is programmed with a specific selection.
b. Similarly we can write an assertion to check input testmux functionality for a case where lets say we want to drive an analog value into the design through input testmux rather coming from another block in the design.

```
// Checking Decode Test mux control signal is Asserted and is in required voltage range
`V_IN_RANGE(EN_TOUT_2p5_BUF,top.vddp5v_hth,top.vddp5v_lth,(top.vddp5v_typ*`vdelta),`ttol,(top.vddp5v_typ*`v
tol),enable_v_assert_trigger,tmux_out_en_1)
// Create a tmux out valid signals based on condition relevant to the design
always @(*)
begin
check_tmux_out_valid = vddp5v_in_range && porb_release && out_of_sd && tmux_out_en;
end
// Digitizing expected tmux out value  for specific mux selection which will be used in Assertion
always (TOUT_MUX_CTRL) begin
…
…
case(tmux_ctrl)
5'd0  : …
5'd16 : tmux_out_ref_2p5 =   `get_abs(V(TOUT_REF_2p5_BUF) - 2.5) < 0.01) ? 1:0;
5'd8  : ….
endcase
// TMUX out Assertion
// CHECKER: when SD is released and TMUX_OUT is enabled with TMUX_OUT ctrl 10000 then TMUX_OUT gets
REF_2p5_BUF with TMUX_OUT_EN<1> high
// psl tmux_out_en_1_assert: assert always ((check_tmux_out_valid  && (tmux_ctrl == 5'd16)) -> (tmux_out_en_1))
@(posedge(mux_ctrl_change_dly));
// psl tmux_out_1_assert: assert always ((check_tmux_out_valid  && (tmux_ctrl == 5'd16) ) -> (tmux_out_1_ref_2p5))
@(posedge(mux_ctrl_change_dly));
```

Fig 6: Example code showing Analog Output Testmux Assertion.

In the above example shown in fig.6, for every combination of testmux selection two assertions which are getting checked are shown.

a. When the tmux out is valid, and if the mux selection is programmed to a specific value then check if the relevant enable signal is activated.
b. When the tmux out is valid, and if the mux selection is programmed to a specific value then check if the mux output is equal to expected value.

Similar approach can be taken to check for the input testmux functionality.

PSL v/s SVA Assertions :

a. The above mentioned verification methodology can be used with SVA assertions, but it is not as straight forward as using PSL assertions.

b. SystemVerilog standard does not allow the presence of continuos domain object. Therefore creating analog expressions is not possible. This capability is possible and comes native to PSL.

c. However SystemVerilog allows the usage of real valued variables and these can be used in the context of SystemVerilog assertions. This is possible with analog value fetch system functions like "cds_get_analog_value".

d. In PSL to obtain the potential of analog singal you can directly bind a vunit to a subckt and use access functions like "V(signal)". We can also use this function directly in the assertion. On contrary this is not possible in SystemVerilog and hence each we want to fetch an analog value we need to use analog fetch system functions and provide the entire design hierarchy path to the required signal.

e. In case fetching current parameter of a signal of interest I(signal) access function does not work for both PSL and SVA. Only way to get this is to use analog fetch system function $CGAV with "flow" as shown in fig.7.

ANALOG VALUE FETCH FOR PSL AND SVA

PSL

Voltage --- Bind Vunit to subckt and use voltage access functions V(). This can be used in PSL Assertions
Current --- `define curr_iztc_500n_0 `get_abs($cgav("<Hierarchial_DUT_path>.IZTC_500n[0]","flow"))

SVA

Analog domain not possible in SVA assertions, use analog fetch functions to get analog values into real variables and use them in assertions.

Voltage --- `define v_refgen_1p2 `get_abs($cgav("<Hierarchial_DUT_path>.IZTC_500n[0]","potential"))
Current --- `define curr_iztc_500n_0 `get_abs($cgav("<Hierarchial_DUT_path>.IZTC_500n[0]","flow"))

Fig.7 Example code showing analog value fetch and usage in PSL and SVA assertions

Methodology and structure for SVA assertions: Since SVA assertions has some limitations related to assertions on analog domain variables, the Vunit based structure explained for PSL is not applicable. If you have to go the SVA route, we need to use SV modules to write SVA assertions and use analog fetch functions to get analog parameters like voltage and current as shown in Fig.7 and assign them to real variables and use them in SVA assertions inside SV module. Fig.8 below shows the topology.

SVA Assertions topology

Top.vams:

```
Module top ()

//instantiate SVA module
sva_module sva_module_i (.....);
 endmodule
```

//SVA assertions module

```
module sva_module(....);

real analog_v, analog_i
analog_v = $cgav("signal_path", "potential");
analog_i = $cgav("signal_path", "flow")
// SVA assertion using above real variables
assert property (@(trig) (cond a) |-> (analog_v > vth));
endmodule
```

Fig.8 Topology for using SVA assertions.

Integration of PSL Assertions into DV environment:

In the incisive or command line flow, these assertions can be pulled in with irun option of "-assert -propfile_vlog" and file name with extension of ".psl". In virtuoso flow, you can enter this option in the ADE state in the additional argument section. Fig.9 shows snapshot of how to pull in psl assertion file into the virtuoso environment.

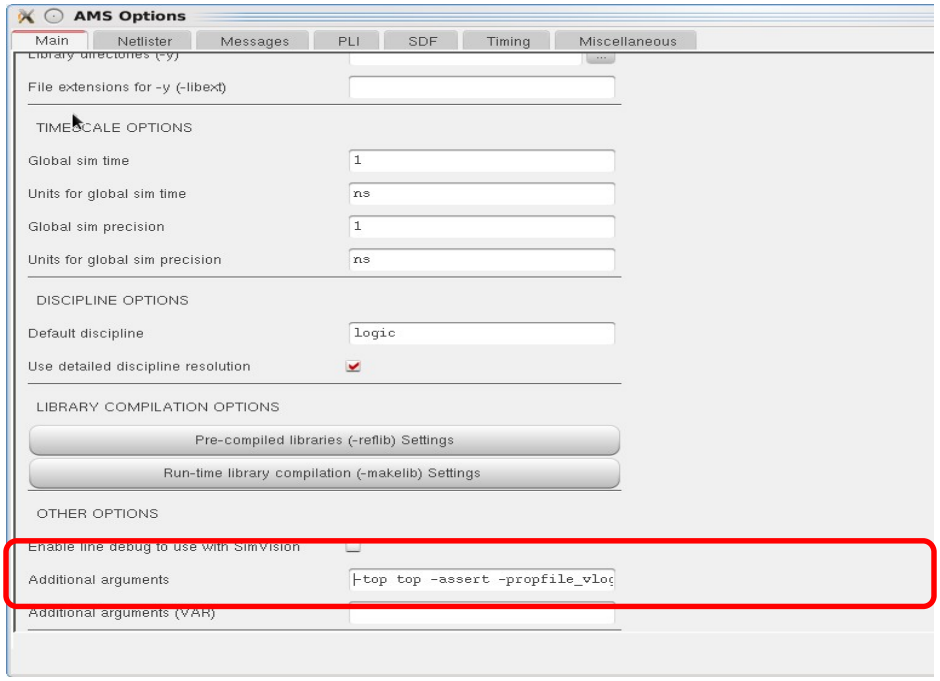

Fig 9: AMS option form in ADE showing arguments to pull in assertion files.

Assertions can be arranged into various vunit files, as per design hierarchy, and since these assertions are bound to the sub-circuit/module, they can be portable along with the design. To pull in multiple assertion files, enter the below commands into a common file and pull in this file with irun or through ADE.

```
// To pull in multiple assertion files with one common file
-assert -propfile_vlog cell1.psl
-assert -propfile_vlog cell2.psl
-assert -propfile_vlog cell2.psl
```

Integration of SVA Assertions into ADE environment: In the incisive or command line flow, these assertions can be pulled in with irun –f top.vams and –top top. In virtuoso flow, you can enter this option in the ADE state in the additional argument section. Fig.10 shows snapshot of how to pull in SVA assertion files into the virtuoso environment. In case you have any PSL assertions in the SVA module you activate them by using a "-assert" option to the irun or enter it in the additional arguments section of the AMS option form.

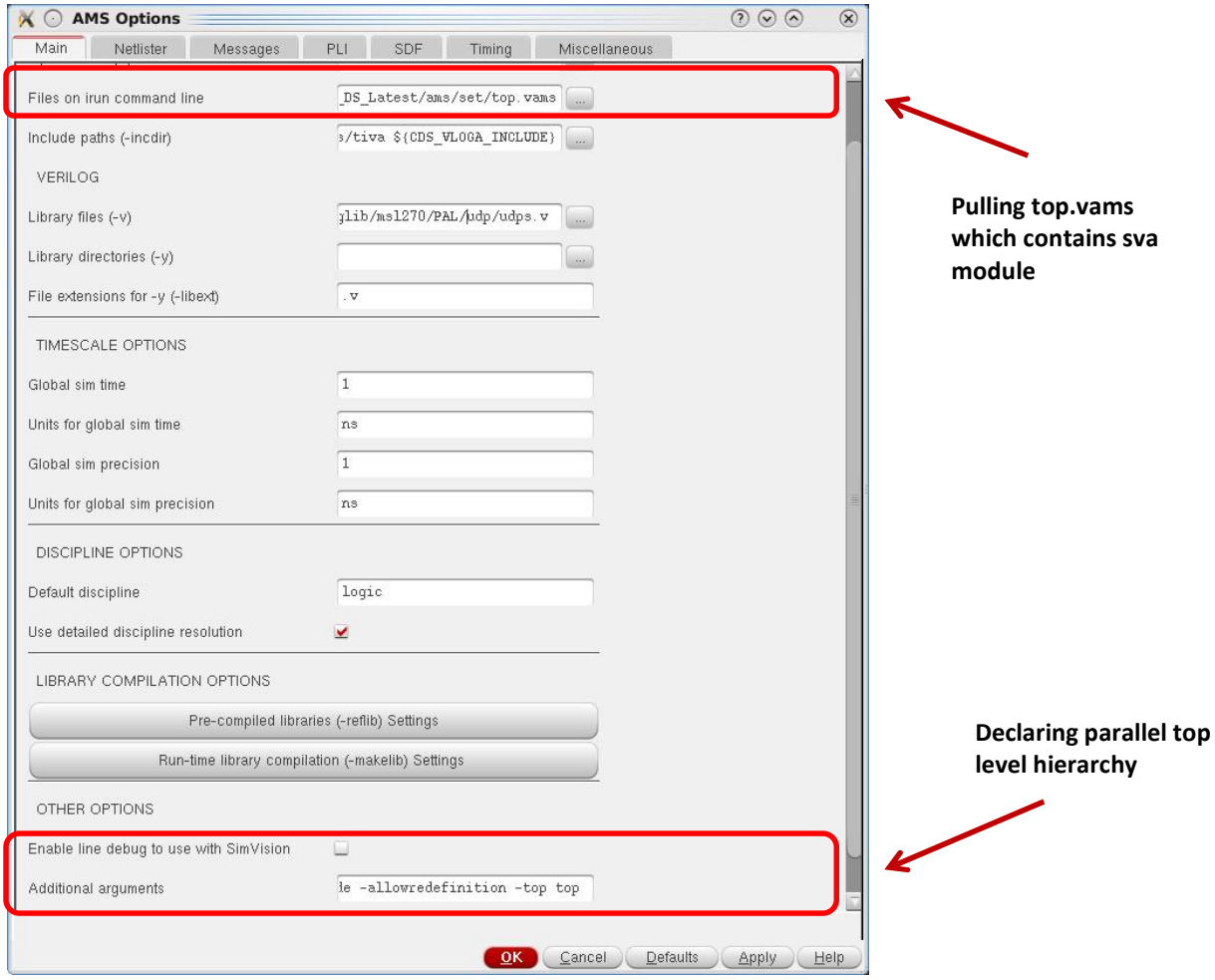

Fig 10: AMS option form in ADE showing arguments to pull in SVA assertion files.

Simulation Results:

Fig10. Shows simulation results of current and voltage assertions along with assertion metrics which is useful in observing how many times each assertion is checked for any possible violation counts.

Current Assertion : For the current assertion each time there is a certain specified change in the value of the IPTAT current, the assertion check triggers to check if the current is within the specified range. Current assertion check uses a layered approach, one to check for actual change which is what the curr_xxx_flag signal is doing, and the other which filters out the flag signal if the out of range duration is very small and acceptable, which is indicated by curr_xxxx_in_range.

Voltage Assertion : For the voltage assertion, the assertion check is triggered whenever ref_1p2 signal undergoes a variation which is more than the specified tolerance which causes the state of ref_1p2_in_range to change, which in turn triggers the assertion and checks for the specification that ref_1p2 should be in expected range once vddp5v is in expected range and any variation later is an error condition.

Also these assertions can be viewed in the assertion browser, which shows the list of all the assertions used, and their stats related pass/fail. Fig11. Shows the snapshot of assertion browser.

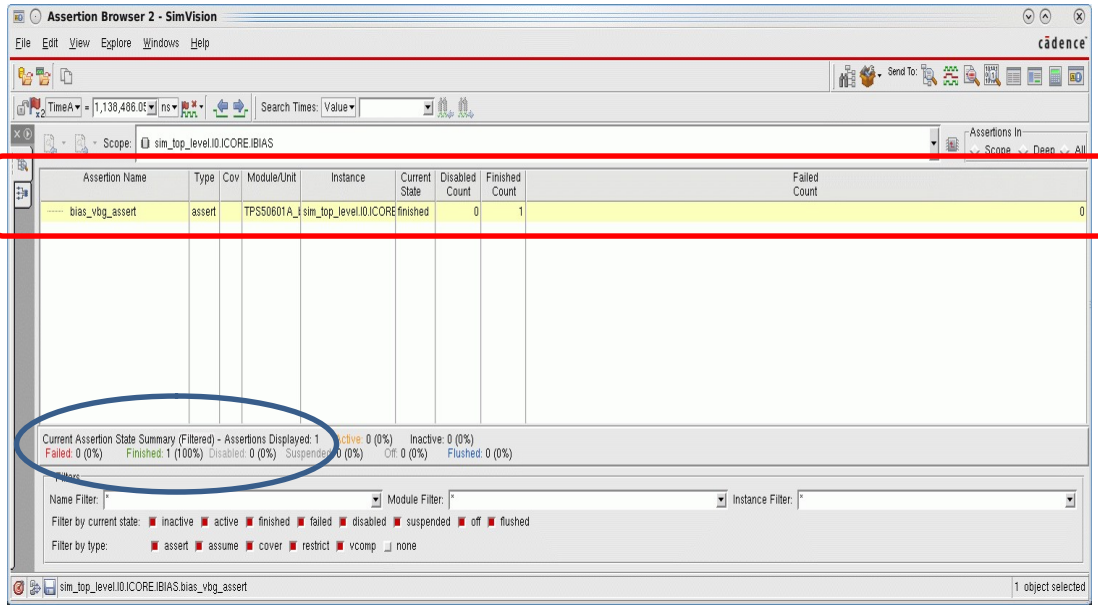

Fig.11: Assertion browser showing assertion statistics.

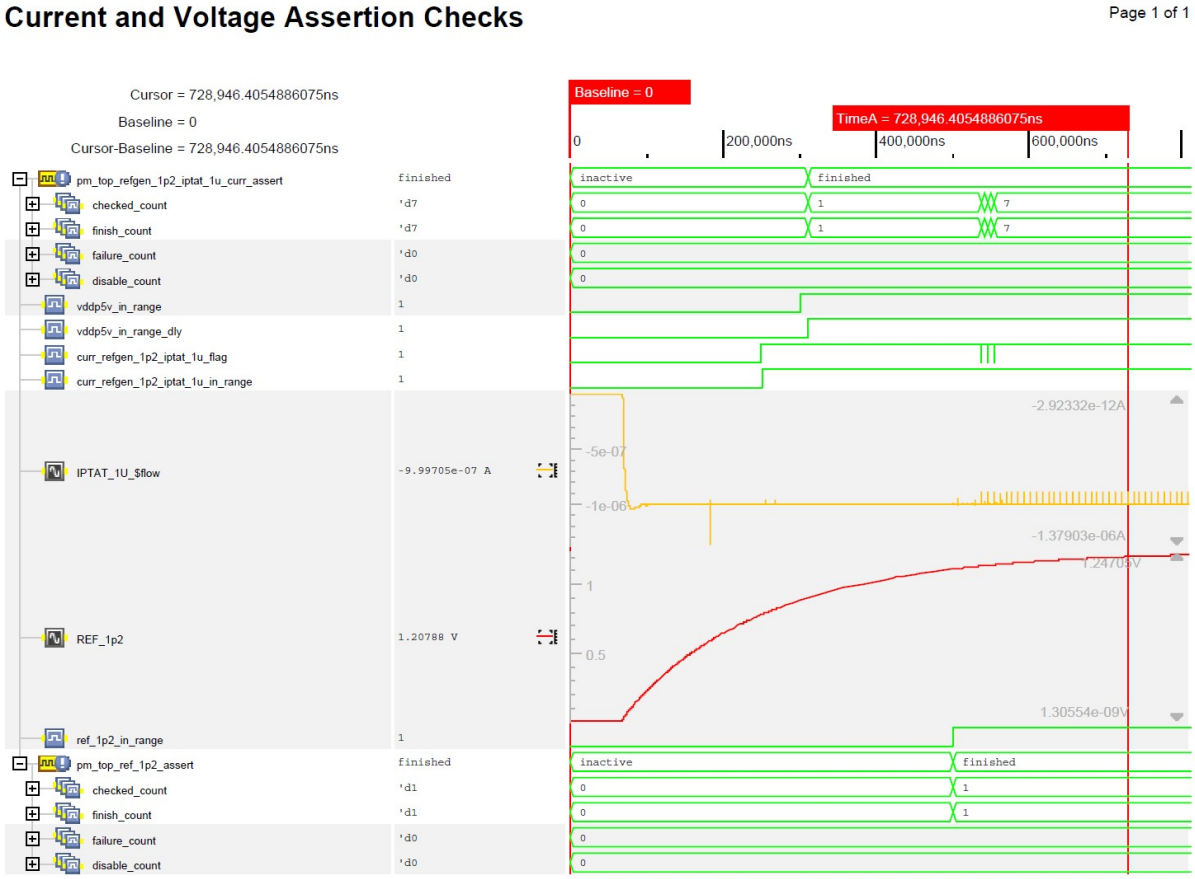

Fig. 11: Simulation results showing current and voltage assertion checks.

Summary: Using the above mentioned strategy, we can successfully implement assertion based verification of analog parameters. This is useful even if the dut is completey an analog design or mixed signal design.Binding assertions to the cell, makes assertions portable and can be effectively re-used along with saving design time in the verification cycle. This implementation is advantageous, over traditional analog checkers where you have to write procedural code using verilogA/Verilog-AMS or using ocean to do post process checking.