Metric Driven Mixed-Signal Verification Methodology and Practices for Complex Mixed-signal ASSPs

Frank Yang, Andy Sha, Morton Zhao
Analog Devices
Beijing, China
Frank.Yang@analog.com
Andy.Sha@analog.com
Morton.Zhao@analog.com

Yanping Sha
Cadence design Systems
Beijing, China
shayp@cadence.com

***Abstract* - Most ASSPs today are mixed-signal systems and this higher level of integration means the verification of these ASSPs is becoming more and more complex. For the 'digital only' SOCs there already exists advanced verification methodologies widely used throughout the industry i.e. VMM, UVM, metric driven verification (MDV). The push in verification is now to extend these advanced verification methodologies to be used in analog/mixed-signal ASSP verification as well.**

A recent ASSP in ADI is an example of this trend. The ASSP was a true mixed signal development, incorporating a Cortex-M3 MCU with analog peripherals including, ADC, VDAC, IDAC and PLL etc. This paper will discuss how we implemented MDV for the mixed-signal ASSP, how we defined verification metrics for analog/mixed-signal blocks and how we built up the mixed-signal constrained random verification environment. Defining verification metrics for analog/mixed-signal blocks is a key common problem for mixed-signal MDV and a lot of the discussions will focus on this topic.

Keywords—Mix-signal Functional Design Verification, Metric Driven Verification (MDV), Assertion Based Verification, Mix-signal ASSP

I. INTRODUCTION

A. Metric Driven Verification Flow

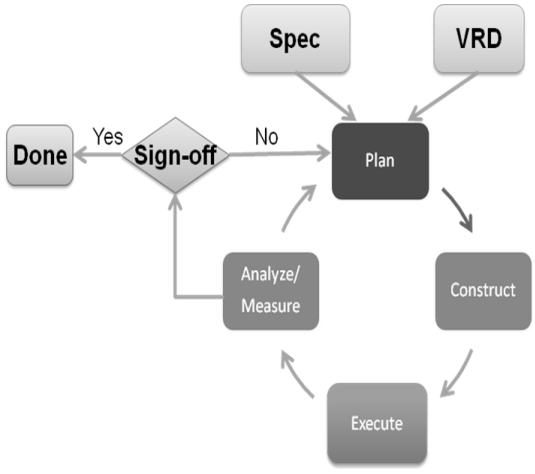

Fig. 1. Metric Driven Verification Flow

Plan: Use Cadence Eplanner to build metric-based executable verification plan according to specification and verification requirement document (VRD).
Construct: Use UVM to build platform and implement verification metrics i.e. assertion and functional coverage.
Execute: Create test cases and run regressions.
Analyze/Measure: Analyze failures and measure metrics.
Sign-off: Verification sign-off based on verification metrics data mapped to vPlan

B. Verification Planning

Metric-driven verification is based on a verification plan. The plan lists the features that need to be verified, what to check for and how to measure coverage [1]. Sources of information for the verification plan are the design specification and verification requirement documents (VRD). The VRD is used to capture the verification requirements of design, application and test engineers. This document will focus on the concerns which design, application and test engineers really want to ensure are covered in the verification process. For example, very detailed design features which are not listed in the design specification. We used Eplanner as the verification plan tool. An example is shown in Fig. 2.

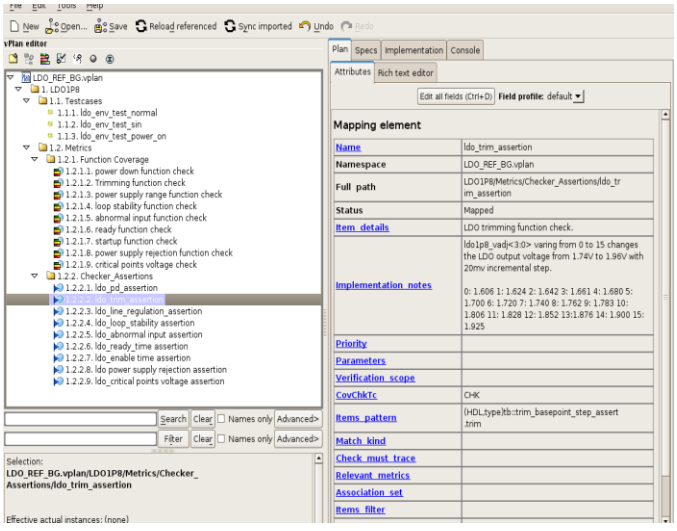

Fig. 2. Example of Verification Plan

C. Verification construct

1) UVM verification enviroment build up

The UVM methodology has succeeded in tackling the hardest verification challenges in digital design. It is a metric-driven approach using coverage directed random stimulus generation, supporting multiple verification languages. It is the methodology of choice to be extended for supporting analog verification. The extended methodology is named UVM-MS [2]. Some simple real-life UVM code is shown in Fig.3.

```
// Integration for UVM-compliant Program block

// Interfaces
`include "mstr_slv_intfs.incl"


module ldo_env_tb_mod; //integration module

import uvm_pkg::*;
`include "uvm_macros.svh"

// ENV+TEST
`include "ldo_env_env.sv"
`include "ldo_env_test.sv"

// All interfaces
   typedef virtual ldo_magt_if    v_if1;
   typedef virtual ldo_sagt_if    v_if2;
   typedef virtual analog_stim_if v_if3;

// Configure TB / Run Test
   initial begin
      uvm_config_db #(v_if1)::set(null,"","mst_if",u_ldo_tb_sv.mst_if);
      uvm_config_db #(v_if2)::set(null,"","slv_if",u_ldo_tb_sv.slv_if);
      uvm_config_db #(v_if3)::set(null,"*.x_analog_stim_agt0","analog_agt_if",
      u_ldo_tb_sv.x_analog_stim_if0);
      uvm_config_db #(v_if3)::set(null,"*.x_analog_stim_agt1","analog_agt_if",
      u_ldo_tb_sv.x_analog_stim_if1);
      run_test();
   end

endmodule: ldo_env_tb_mod
```

Fig. 3. UVM example

2) Analog/MS verification metric definition and implementation

We defined the following types of analog/mixed-signal verification metrics.

a) Real number analog assertions

The following table shows the features which are suitable to use real number analog assertions to define metrics.

TABLE I. EXAMPLES OF REAL NUMBER ANALOG ASSERTION

Features	Detailed Features and Examples	
Timing	Settling time, none-overlap signals timing	
Digital controlled analog trimming	Voltage trimming	Reference/LDO/POR trip point trimming
	Current trimming	Bias current trimming
	Clock frequency trimming	VCO/Oscillator trimming
Digital assisted analog calibration	ADC offset/gain calibration	
	DAC offset/gain calibration	
Algorithm	ADC chop, average, redundant, dither…	
Critical signal monitors	Monitor power supply, reference and bias	

b) Real Number Functional Coverage

The following table shows the features which are suitable to use real number function coverage to define metrics.

TABLE II. EXAMPLES OF REAL NUMBER FUNCTIONAL COVERAGE

Features	Detailed Features and Examples
Voltage range	ADC input voltage range
	DAC output voltage range
	Voltage reference range
Current range	Bias current range
Clock frequency range	PLL VCO clock frequency range
	PLL input reference clock range

c) Memery mapped register function coverage

Function coverage of memory mapped register is defined to cover the function of each bit of memory mapped register.

d) Toggle coverage of analog/digital interface signals

The toggle coverage of analog/digital interface signals is defined to cover the different control sequences.

e) Code coverage

Code coverage of RTL in analog control blocks.

II. REAL LIFE CASE STUDY: LDO

A. LDO specification

In our project there was a low drop-out regulator (LDO) which provided a 1.8 V supply to the MCU chip, including all 1.8 V analog blocks and all digital blocks, such as CORTEX-M3, flash, SRAM etc. The LDO can output a maximum current of 100 mA. A 4-bit trim was employed to tune the LDO output level.

TABLE III. LDO ELECTRONIC SPECIFICATION

Parameter	Test Condition	Min.	Typ.	Max	Unit
Input operating voltage		2.8	3.3	3.7	V
Output current			100		mA
Output voltage	Trim code = 9	1.74	1.80	1.96	V
Trim step for output voltage			0.02		V
Quiescent Current			200		μA
Power supply rejection	@ 1 kHz		-50		dB
Phase margin		60			Deg
Gain margin		10			dB

TABLE IV. LDO MEMORY MAPPED REGISTER DEFINITION

Bits	Bit Name	Description	Reset	Access
[4:1]	LDO_VREG_ADJ	Trim bits to adjust the LDO output voltage	0x9	RW
0	LDO_PD	LDO Power down signal (active high).	0x0	RW

B. *Block-box verification approach*

Black-box verification refers to the technique of verification system with no knowledge of the internals of the DUT. Black box testbenches do not have access to the source code of the DUT, and are oblivious of the DUT architecture. A black box testbench, typically, interacts with a system through a user interface by providing inputs and examining outputs, without knowing where and how the inputs were operated upon. In black box verification, the target DUT is exercised over a range of inputs, and the outputs are observed for correctness. How those outputs are generated or what is inside the box doesn't matter [3]. The black box symbol for the LDO is shown in Fig.4.

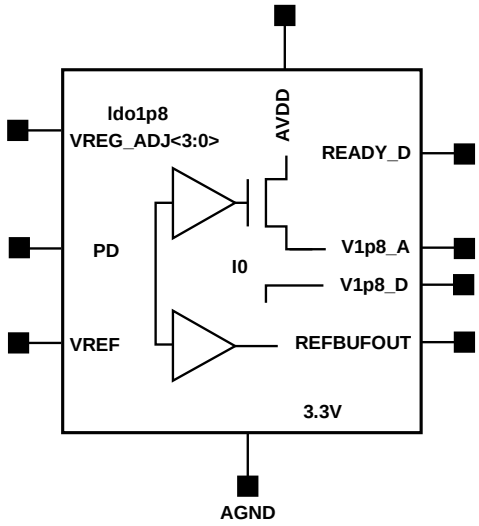

Fig. 4. Black-box symbol of LDO

We developed a black box reference model to auto-check the LDO functionality. From functionality viewpoint LDO output was the function of voltage reference vref and trim word vreg_adj. The reference model code is shown in Fig.5.

```
function ldo_sagt_item ldo_sb::ref_model_process(ldo_magt_item tr);

    if(tr == null)
        `uvm_fatal("LDO_SB", "Master ITEM is EMPTY")
    else
    begin
        ref_model_process = new("ref_model_process");
        if(tr.pd == 1)
        begin
            ref_model_process.v1p8      = 0.0;
            ref_model_process.refbufout = 0.55;
        end
        else
        begin
            ref_model_process.v1p8      = tr.vref*(1.96+0.0239*
                                          []((real'(tr.vreg_adj)) - 10.0));
            ref_model_process.refbufout = tr.vref/1.314;
        end
    end

endfunction
```

Fig. 5. LDO reference model

Based on the LDO specification and peer brainstorming we defined the following black box verification metrics.

1) *Real number analog assertion checks*

For the analog/mixed-signal verification we're often asked to check the voltage value at certain condition. One real number analog assertion v_checker was developed for this purpose. It will be used in the later sections. The v_checker code is shown in Fig.6.

```
// Real number analog assertion for voltage checker
module v_checker
       #(
       parameter base_point = 3.02, // expected value
       parameter tolerance  = 0.01, // Tolerance by relative value
       parameter delay_check = 10   // Delay n sample clocks to check
       )
       (
        input   real    compare_value, // Measured voltage value
        input           sample_clk,    // sample clock
        input           disable_check, // Disable check condition
        input           compare_enable // Enable comparision
        );

  property value_check_assert;
   @(posedge sample_clk) disable iff(disable_check)
    (compare_enable) |-> ## delay_check
    (compare_value < base_point * (1 + tolerance))
    && (compare_value > base_point *(1 - tolerance));
    endproperty

  a_value_checker: assert property (value_check_assert)
      else
      $display("Assertion ERROR: the value comparasion\
      has been failed at the time @%t",$time);

  c_value_check_assert: cover property (value_check_assert);

endmodule
```

Fig. 6. Voltage checker

2) *Power supply rejection check*

Real number analog assertions can be used not only for functionality checks but also for some analog performance checks. One example is the LDO power supply rejection check. According to the LDO electronic specification, the PSR was -50db, and we have shown the code for the real number analog assertion in Fig.7.

```
// Real number analog assertion: LDO power supply rejection performance check
module psr_chk_db
 #(
 // LDO PSR spec: -50 db
   parameter psr_spec_db = -50.00,
 // Expected voltage output of the LDO: 1.8v
   parameter base_point = 1.8,
 // Amplitude of the sine wave noise: +/-50 mv
   parameter power_noise  = 0.05,
  // Calculated torerance
   parameter tolerance = ( power_noise* 10**(psr_spec_db/20.0))/base_point
  )
  (
    input real v1p8_a,      // the voltage output of the LDO
    input      ldo_pd,      // power down LDO
    input      sample_clk   // Sample clock
   );

v_checker
 #(.base_point(base_point),
   .tolerance(tolerance),
   .delay_check(0)
  )
 u_v_checker
 (.compare_value(v1p8_a),
  .sample_clk(sample_clk),
  .disable_check(ldo_pd),
  .compare_enable(1'b1)
  );

endmodule
```

Fig. 7. Top-level PSR analog assertion code

The simulation waveform for the PSR check is shown in Fig.8.

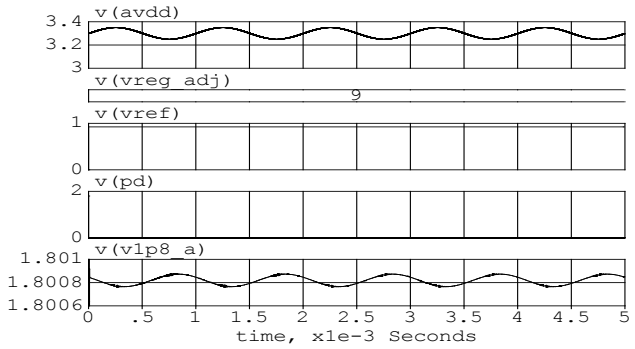

Fig. 8. Simulation waveform of LDO PSR check

The target trim value check

The target trim value should be in the center of trim range. If the target trim value is close to the boundary of trim range there is a risk that some devices cannot be trimmed to the target value due to process variation. The 4 bits trim word ldo_vadj was used to trim the LDO voltage to a target value of 1.8 V. The following top-level analog assertion code was used to check if the target value 1.8 V was at the center of the trim range.

```
// Assertion check: Check if the targe trim value is around at
// the central of the trim range
trim_balance_point
#  (
    .TRIM_WORD_WIDTH (4),      // 4 bit trim word
    .target_value(1.8),        // Target trim value is 1.8v
    .trim_word_max_value(15), // Max trim word is 15
    .trim_word_min_value(0),   // Min trim word is 0
    .tolerance (0.05)          // 5% tolerance
   )
u_trim_balance_point
    (
    .voltage(ldo_out),
    .disable_check(pd || (lp_ref < 0.88) || (vdd3p3_ldo < 2.8)),
    .sample_clk(sample_clk),
    .trim_word(ldo_vadj)
_   );
```

Fig. 9. LDO target trim value check

Abnormal condition check

If the LDO was enabled but either the power or reference supplies were not available then the correct LDO output would not be provided. Analog assertion code for this check is shown in Fig.10 and Fig.11.

```
module abn_input_check
    #(parameter delay_abn_input_check = 20)
    (
        input sample_clk,
        input disable_check,
        input lost_input,    // when some input lost,
                             // the outcome will generate.
        input outcome_check
    );

property abn_input_ast;

@(posedge sample_clk) disable iff (disable_check)
    lost_input |-> ##delay_abn_input_check outcome_check ;
endproperty

 a_abn_input_ast: assert property (abn_input_ast)
      else
      $display("Assertion ERROR: the abn_input_ast has been \
      failed at the time @%t",$time);

   cover_abn_input_ast: cover property (abn_input_ast);

endmodule
```

Fig. 10. LDO abnormal condition check #1

```
//this assertion is to check the abnormal input function check,
//either of the vdd3p3_ldo,lp_ref are lack of ,
//the output voltage should be 0

 abn_input_check
    #(.delay_abn_input_check(2))
 u_abn_input_check
    (
      .sample_clk(sample_clk),
      .disable_check((vdd3p3_ldo >2.8) && (lp_ref > 0.88)),
      .lost_input((vdd3p3_ldo <0.5) || (lp_ref <0.5)),
      .outcome_check(ldo_out < 0.7)
    );
```

Fig. 11. LDO abnormal condition check #2

3) Real number functional coverage

Power supply voltage range

We defined the real number functional coverage to make sure the minimum value, typical value, maximum value and other values of the power supply were covered by our verification. The real number functional coverage for the power supply is shown in Fig.12.

```
covergroup real_avdd_bin_cg; // Function Coverage for LDO power supply range
  option.per_instance = 1;
  ldo_avdd_range : coverpoint avdd_r {
    bins Avdd_Min      = {2.8};            // Min value
    bins Avdd_Typ      = {3.3};            // Typ Value
    bins Avdd_Max      = {3.7};            // Max value
    bins Avdd_range1   = {[2.8  : 3.0  ]}; // 2.8-3.0
    bins Avdd_range2   = {[3.0  : 3.1  ]}; // 3.0-3.1
    bins Avdd_range3   = {[3.1  : 3.3  ]}; // 3.1-3.3
    bins Avdd_range4   = {[3.3  : 3.5  ]}; // 3.3-3.5
    bins Avdd_range11  = {[3.5  : 3.7  ]}; // 3.5-3.7
  }
endgroup : real_avdd_bin_cg
```

Fig. 12. Real number function coverage for LDO power supply

Voltage reference range

We defined the real number function coverage to make sure the minimum value, typical value, maximum value and other values of voltage reference were covered by our verification. The real number function coverage for voltage reference is shown in Fig.13.

```
real vref_r;

covergroup real_vref_bin_cg; // Function Coveage for LDO Vref range
  option.per_instance = 1;
  vref_in_range : coverpoint vref_r {
    bins Vref_Min      = {0.89};            // Min value
    bins Vref_Typ      = {0.92};            // Typ Value
    bins Vref_Max      = {0.962};           // Max value
    bins Vref_range1   = {[0.89  : 0.92 ]}; // 0.89-0.92
    bins Vref_range2   = {[0.92  : 0.962]}; // 0.92-0.962
  }
endgroup : real_vref_bin_cg
```

Fig. 13. Real number function coverage for LDO voltage reference

4) Function coverage

Virtual MMR bits for load configuration

The load conditions are very important for LDO verification. In the test bench we defined virtual MMR bits load_cfg[1:0] to configure the LDO load conditions. If load_cfg = 2'b00 a 1.8K Ohm resistor load will be selected.

If load_cfg = 2'b01 a 180 Ohm resistor load will be selected. If load_cfg = 2'b10 a 36 Ohm resistor load will be selected. If load_cfg = 2'b11 a 18 Ohm resistor load will be selected. In order to make sure all the load configurations were covered during verification we defined the following functional coverage.

```
covergroup cg_mmr_trans @(cg_mmr_event);
    option.per_instance = 1;
    load_cov: coverpoint load_cfg {
                bins low    = {2'b00}; // 1 mA - 1.8K Ohm resistor Load
                bins normal = {2'b01}; // 10mA - 180 Ohm resistor load
                bins high   = {2'b10}; // 50mA - 36 Ohm resistor load
                bins V_high = {2'b11}; // 100mA - 18 Ohm resistor load
               }
 endgroup: cg_mmr_trans
```

Fig. 14. Function coverage for LDO load conditions

MMR bits coverage and control signal toggle coverage

Functional coverage for MMR bits was used to make sure each function defined with MMR bits was covered during verification. Toggle coverage of control signals was used to cover different control sequences. The code is shown in Fig.15.

```
covergroup cg_mmr_trans @(cov_mmr_event);
    option.per_instance = 1;
    option.name = "ldo_stim_cov";
    pd:    coverpoint tr.pd    {
                                  bins zero = {0};
                                  bins one  = {1};
                                  bins z2o  = (0=>1);
                                  bins o2z  = (1=>0);
                               }
    vreg: coverpoint tr.vreg_adj;

endgroup: cg_mmr_trans
```

Fig. 15. Function coverage for MMR and toggle coverage for control signals

C. White-box verification approach

In white-box verification an internal perspective of design are used to design test cases, and the test bench has access to internal structures of design. The block diagram for the LDO with internal structure is showed in Fig.16.

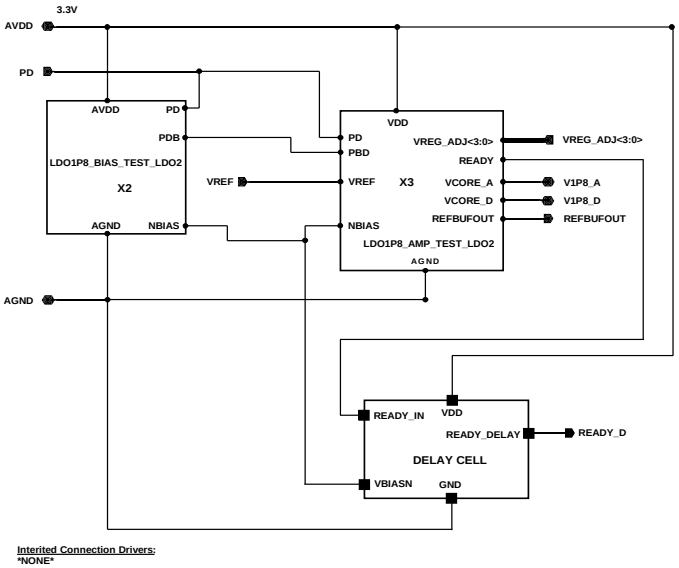

Fig. 16. Block diagram of the LDO with internal structures

Internal bias check

Within the LDO there was a bias generation block to provide ~10 µA bias and we needed to verify that the bias current was always in the correct range. In the test bench the current signal was converted to a voltage signal so we could use the v_checker assertion described earlier to check the bias current. The code is shown in Fig.17.

```
// This assertion check LDO nbias current.
v_checker
#(
  .base_point(10), // Expected nbias is 10uA.
  .tolerance(0.1), // +/- 10% Tolerance
  .delay_check(15)
  )
u_v_checker
 (
   .compare_value(nbias_i2v),// nbias value after current to volteage conversion
   .sample_clk(sample_clk),
   .disable_check(ldo_pd),
   .compare_enable((lp_ref > 0.88) && (vdd3p3_ldo > 2.8))
 );
```

Fig. 17. Bias current check

D. Constrained random testbench build

We built the constrained random test bench to speed-up the verification closure. Both analog and digital signals can be randomized. For example the power supply was randomized from 2.8 V to 3.7 V, reference voltage was randomized from 0.892 V to 0.947 V and the power down signal was randomized with 85% on and 15% off. The code is shown in Fig.18.

```
// ------------------------------------------------
// generate random avdd/vref/pd/trim to check if
// the output is the same with golden model
// ------------------------------------------------
class sequence_1 extends base_sequence;    // normal
  byte sa;
  `uvm_object_utils(sequence_1)
  `uvm_declare_p_sequencer(ldo_magt_sqr)
  `uvm_add_to_seq_lib(sequence_1, ldo_magt_sqr_sequence_library)
  function new(string name = "seq_1");
    super.new(name);
  endfunction:new
  virtual task body();
    #400ns;
    repeat(20) begin
      `uvm_do_with(req, {
                          // Power Supply is 2.8V~3.7V
                          avdd inside {[2.8:3.7]};
                          // Reference Voltage is 0.892V~0.947V
                          vref inside {[0.892:0.947]};
                          // Ground is 0.0V
                          agnd == 0.0;
                          // Power 85%---ON  15%---OFF
                          pd dist {0:=85, 1:=15};
                        } );
      #1500us;
    end
  endtask
endclass
```

Fig. 18. Code for constrained random test bench

The simulation waveform is shown in Fig.19.

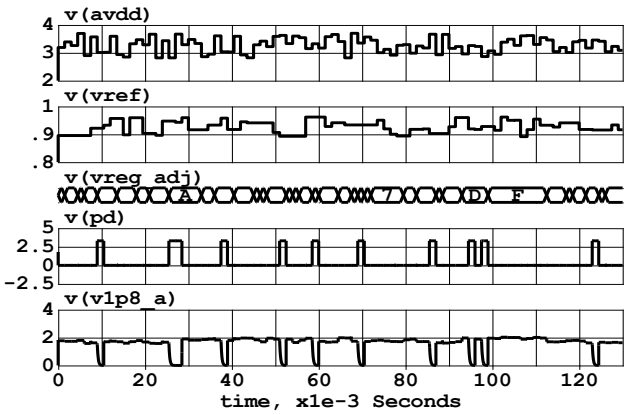

Fig. 19. Simulation waveform for constained random

E. Verification sign-off

We used Emanager to manage the regressions and metric collection. The collected metric data was mapped onto planned metrics in vPlan, and displayed in a html report of which a screenshot is shown in Fig.20.

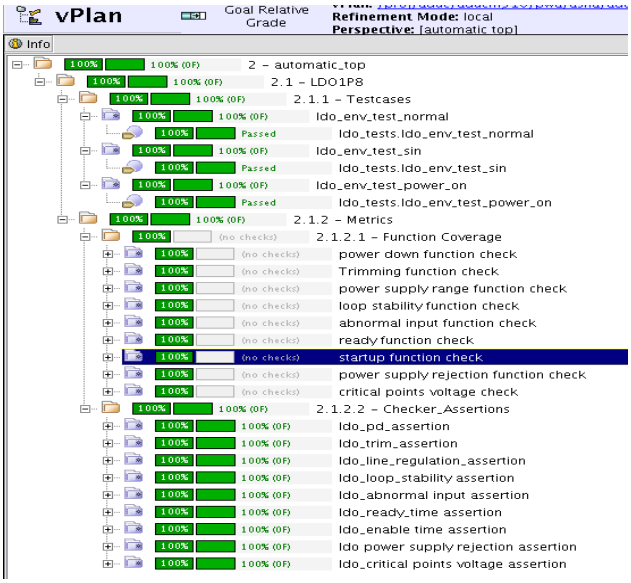

Fig. 20. Metric data mapped back to vPlan

III. CHIP-LEVEL VERIFICATION

A. Chip-level connection verification

1) Level shifter check

Level shifters should be placed when the signals cross the power domains within the design. The LDO power down and trim word signals passed from the 1.8 V digital power domain to the 3.3 V analog power domain. So when we did top-level verification we needed to check if level shifters existed. We could use a simple voltage checker to do this task. The code is shown in Fig.21.

```
// This assertion check if there are level shifters for ldo control signals
// from 1.8 v digital power domain to 3.3 v analog power domain
module ldo_level_shifter_check (
    input real  ldo_pd_a,
    input real  ldo_vadj_0,
    input real  ldo_vadj_1,
    input real  ldo_vadj_2,
    input real  ldo_vadj_3,
    input real  vdd3p3_ldo,
    input       sample_clk
    );

v_checker
#( .base_point(3.3), // 3.3v power domain
  .tolerance(0.1),  // +/- 10% Tolerance
  .delay_check(15))
u_level_shifter_chk_0
 ( .compare_value(ldo_pd_a),// ldo_pd should be in 3.3 power domain
   .sample_clk(sample_clk),
   .disable_check(ldo_pd_a < 1.6),
   .compare_enable(vdd3p3_ldo > 3.1));

v_checker
#( .base_point(3.3), // 3.3v power domain
  .tolerance(0.1),  // +/- 10% Tolerance
  .delay_check(15))
u_level_shifter_chk_1
 ( .compare_value(ldo_vadj_0),// lod_vadj[0] should be in 3.3 power domain
   .sample_clk(sample_clk),
   .disable_check(ldo_vadj_0 < 1.6),
   .compare_enable(vdd3p3_ldo > 3.1));
```

Fig. 21. Level shifter check

2) Critical signal connection check

When we did the top-level verification we needed to check that all critical signal connections, that is, the power supply and reference connections were made correctly. Our code for these checks is shown in Fig.22.

```
// This assertion check if connections for LDO power supply and reference
module ldo_connection_check (
    input ldo_pd,
    input real ldo_vref,
    input real vdd3p3_ldo,
    input sample_clk
    );

v_checker
#( .base_point(0.92),// 0.92v LDO vref
  .tolerance(0.03),  // +/- 3% Tolerance
  .delay_check(15))  // Check after 15 sample clocks
u_connection_chk_0
 ( .compare_value(ldo_vref), // LDO reference connection check
   .sample_clk(sample_clk),
   .disable_check(ldo_pd),
   .compare_enable(1'b1));

v_checker
#( .base_point(3.3),// 3.3v LDO power supply
  .tolerance(0.1),  // +/-10% Tolerance
  .delay_check(15))  // Check after 15 sample clocks
u_connection_chk_1
 ( .compare_value(vdd3p3_ldo), // LDO power supply cronnection check
   .sample_clk(sample_clk),
   .disable_check(ldo_pd),
   .compare_enable(1'b1));
```

Fig. 22. Power supply and reference connections check

B. Metrics reuse for model validation

A key component in chip-level mixed-signal verification is modeling. In order to make sure the analog models were correct we needed to run model validation. All the black-box metrics which were used for analog block-level verification could be reused for model validation. The white-box metrics could be or could not be reused for module validation depending on the abstraction level of the model.

C. Auto generation of MMR functional coverage

We developed scripts to automatically generate all the MMR functional coverage codes. One piece of MMR

functionality coverage code for ADC control register is shown in Fig.23.

```
class reg_ADC_CORE_ADCCON extends uvm_reg;
        rand uvm_reg_field C_TYPE;
        rand uvm_reg_field CNV_DMA;
        rand uvm_reg_field SEQ_DMA;
        rand uvm_reg_field Restart_ADC;
        rand uvm_reg_field REFB_PUP;
        rand uvm_reg_field PUP;
        rand uvm_reg_field SOFT_RESET;

        constraint C_TYPE_valid       {  }
        constraint CNV_DMA_valid      {  }
        constraint SEQ_DMA_valid      {  }
        constraint Restart_ADC_valid  {  }
        constraint REFB_PUP_valid     {  }
        constraint PUP_valid          {  }
        constraint SOFT_RESET_valid   {  }

        covergroup cg_vals;
          option.per_instance = 1;
          C_TYPE: coverpoint C_TYPE.value[2:0];
          CNV_DMA: coverpoint CNV_DMA.value[0:0];
          SEQ_DMA: coverpoint SEQ_DMA.value[0:0];
          Restart_ADC: coverpoint Restart_ADC.value[0:0];
          REFB_PUP: coverpoint REFB_PUP.value[0:0];
          PUP: coverpoint PUP.value[0:0];
          SOFT_RESET: coverpoint SOFT_RESET.value[5:0];
        endgroup

        function new(string name = "ADCCON");
            super.new(.name(name), .n_bits(16), .has_coverage(UVM_CVR_FIELD_VALS));
            if (has_coverage(UVM_CVR_FIELD_VALS)) begin
              cg_vals = new();
            end
        endfunction: new
```

Fig. 23. Auto-generated MMR function coverage

D. Power-up sequence verification

For multi-power domain systems the power up sequence is very critical. We defined 2 timing parameters for power ramp up: t1, the start time of the power ramp up, and t2, the power ramp up time. There were 2 power supplies avdd1 and avdd2 in our chip. By defining the functional coverage which is shown in Fig.24 we could make sure that all the power up sequence scenarios were covered by verification.

```
covergroup ldo_power_on_seq;

    option.per_instance = 1;
    option.name = "ldo_power_on_seq";
    avdd1_t1: coverpoint pitem1.t1 {
                                        bins avdd1_t1_A = {[1000.0:2000.0]};
                                        bins avdd1_t1_B = {[2000.0:3000.0]};
                                        bins avdd1_t1_C = {[3000.0:4000.0]};
                                        bins avdd1_t1_D = {[4000.0:5000.0]};
                                    }
    avdd2_t1: coverpoint pitem2.t1 {
                                        bins avdd2_t1_A = {[1000.0:2000.0]};
                                        bins avdd2_t1_B = {[2000.0:3000.0]};
                                        bins avdd2_t1_C = {[3000.0:4000.0]};
                                        bins avdd2_t1_D = {[4000.0:5000.0]};
                                    }
    avdd1_t2: coverpoint pitem1.t2 {
                                        bins avdd1_t2_A = {[1000.0:2000.0]};
                                        bins avdd1_t2_B = {[2000.0:3000.0]};
                                        bins avdd1_t2_C = {[3000.0:4000.0]};
                                        bins avdd1_t2_D = {[4000.0:5000.0]};
                                    }
    avdd2_t2: coverpoint pitem2.t2 {
                                        bins avdd2_t2_A = {[1000.0:2000.0]};
                                        bins avdd2_t2_B = {[2000.0:3000.0]};
                                        bins avdd2_t2_C = {[3000.0:4000.0]};
                                        bins avdd2_t2_D = {[4000.0:5000.0]};
                                    }
    power_on: cross avdd1_t1, avdd2_t1, avdd1_t2, avdd2_t2;

endgroup
```

Fig. 24. Function coverage for power up sequence

IV. CONCLUSIONS

Implementing these advanced metric driven mixed-signal verification techniques on our recent ASSP development was a great success and resulted in first pass silicon success. The initial revision of silicon was successfully sampled to our customers.

REFERENCES

[1] Jess Chen, Michael Henrie, Monter F. Mar, Mladen Nizic, Mixed-Signal Methodology Guide, ISBN 978-1-300-03520-6

[2] Bishnupriya B., John D., Gary H., Nick H., Yaron K., Neyaz K., Zeev K., Efrat S., 2012, Advanced Verification Topics, Cadence Design Systems, ISBN 978-1-105-11375-8

[3] http://testbench.in/TB_34_WHITE_GRAY_BLACK_BOX.html