2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Functional Verification of Analog Devices modeled using SV-RNM

Mariam Maurice

Siemens Industry Software Inc.

SIEMENS

accellera
SYSTEMS INITIATIVE

Agenda

Real Number Modelling

- Pros
- Increase Accuracy
- Design Contributions

Functional Verification

- Contributions
- Constrained Random Verification (CRV)
  - Randomization as modeling mismatch effect
  - Writing constraints for the mismatch effect
  - Randomization of Input Real/logic signals
- Assertions/Checkers
  - Relation between I/O ports

Agenda

UVM

- Interfaces
- Sequence Generation
  - Randomization of Input ports
  - Randomization of Internal Signals
- Monitors
- Subscribers
- Scoreboard

Check on Linearity

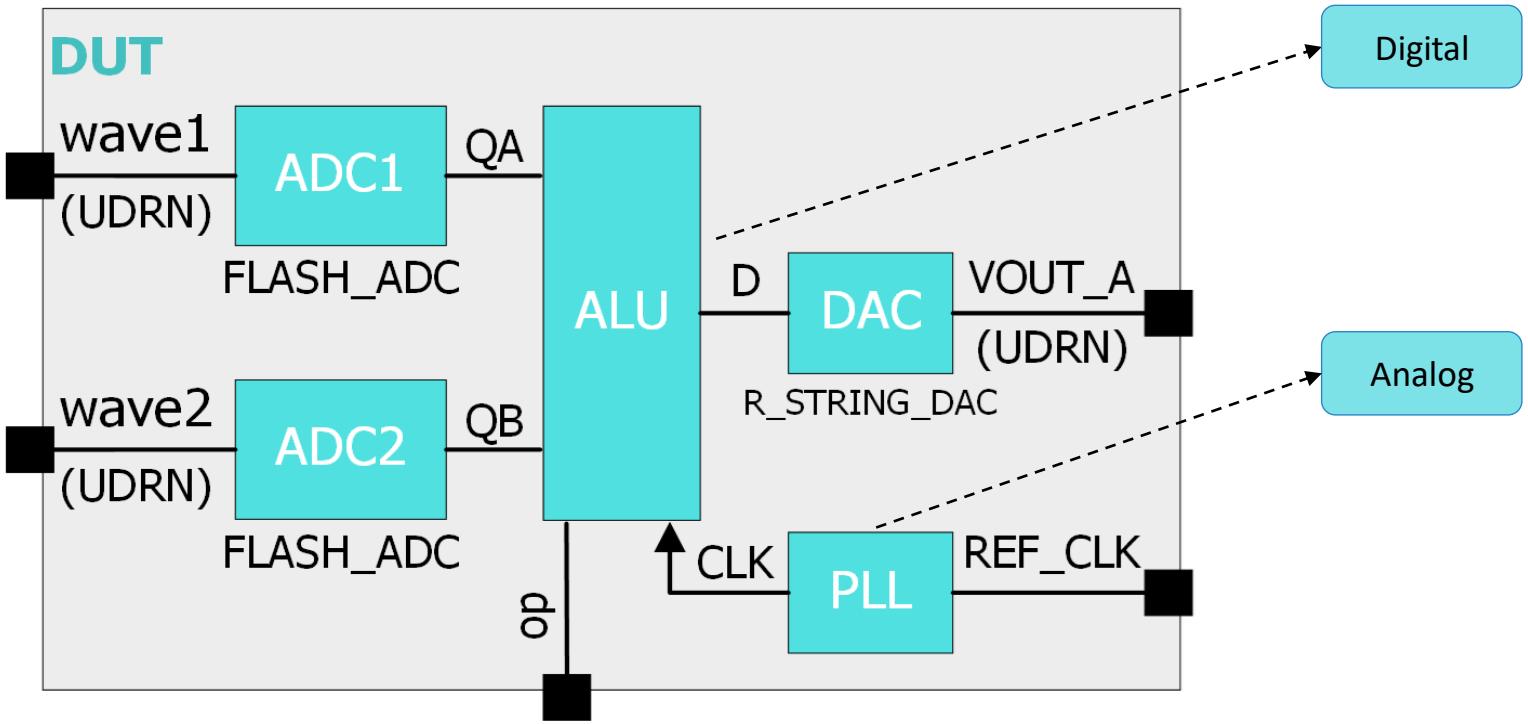
- INL/DNL

Results

- Scoreboard

Real Number Modeling – Why?

- Waiting for transistor level to be finished could consume a lot of time for digital verification engineers, to ensure that both systems (the analog, and the digital) will work functionality well when they are connected together.



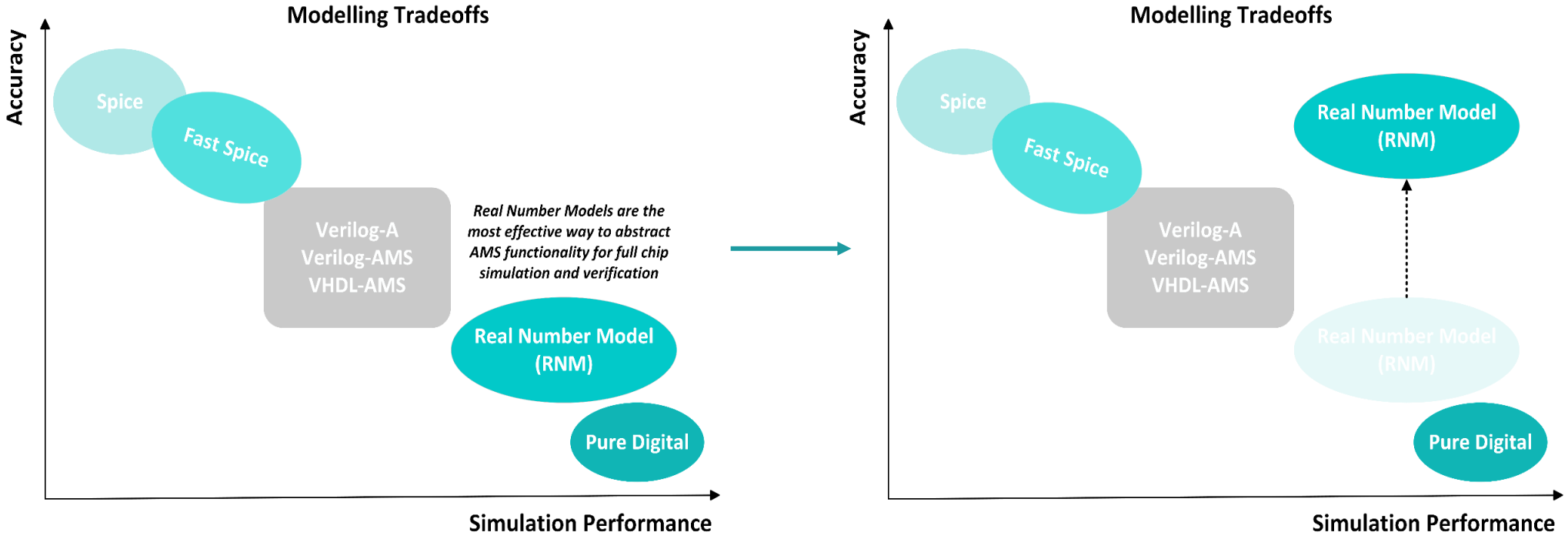
Real Number Modeling – Pros

- Real Number Modelling (RNM) is the process of modelling the behavior of an analog circuit as a discrete real data so it's a signal flow-based modelling approach.

Speed	• 10X – 1000X faster than SPICE • 10X – 100X faster than Verilog-AMS electrical
Accuracy	• More accurate than a pure logic model (0,1) • RNM constructs to increase accuracy
Top Down	• Start verification without finished schematics • Reuse model
Ease of use	• Single Simulator • Leverage existing digital tools & flows

Real Number Modeling – Increase Accuracy

- The previous research provides the increase in the modelling accuracy of analog devices (especially for the ADC/DAC/PLL) with maintaining high simulation performance as an event-driven simulator is used.

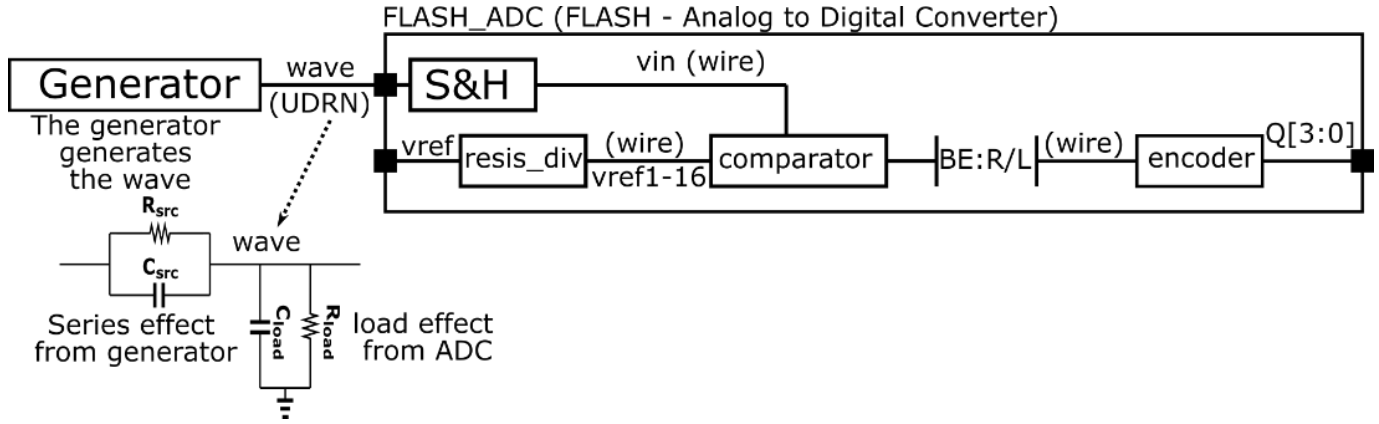


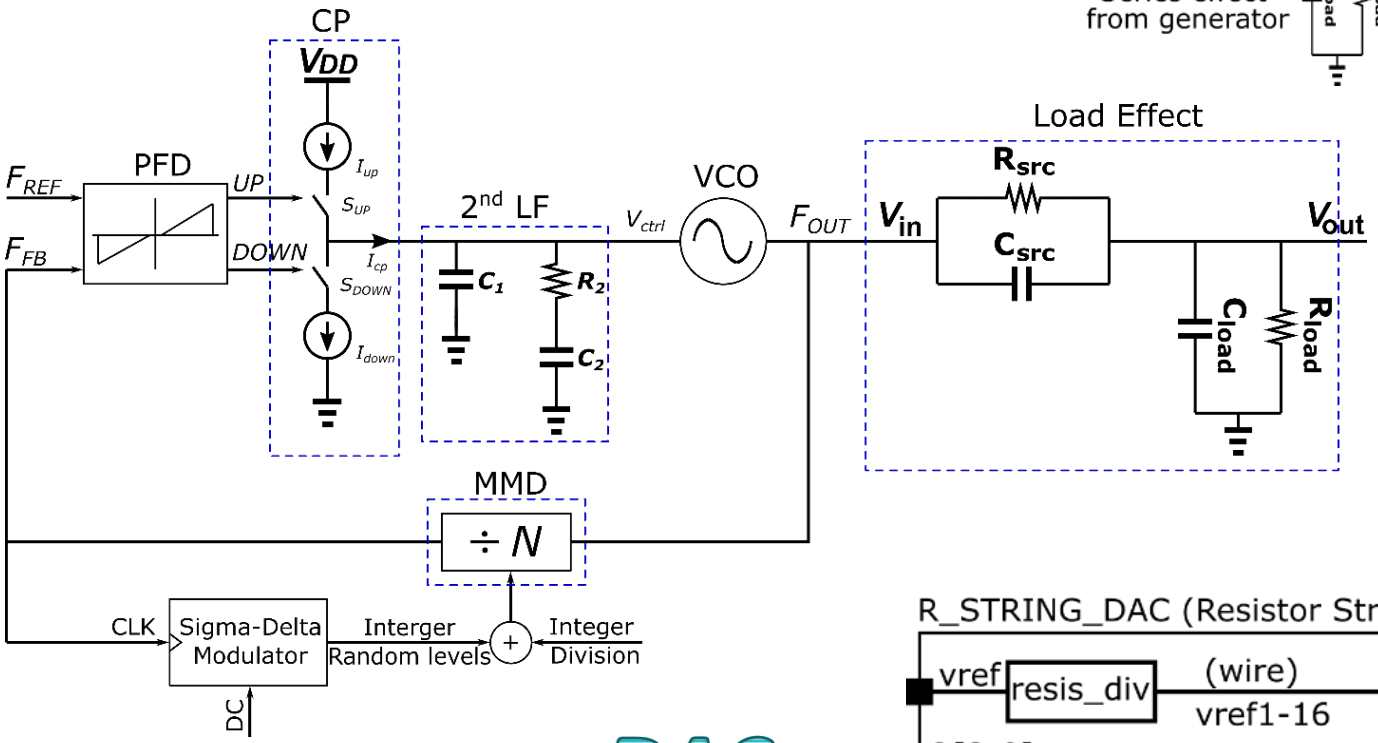
Real Number Modeling – Design Contributions

- SystemVerilog (SV) is chosen as it provides a lot of datatypes that helps in constructing the Real Number Models and these RNM constructs increase the modelling accuracy:
  - Real datatype
    - "Model Quantization noise effect"
    - "Calculate INL/DNL of ADC/DAC"
  - User-Defined nets
    - "Model 2nd order Loop filter using PWL technique"
  - User-Defined resolved nets
    - "Model Load effect"
  - Class datatypes (Object Oriented Programming 'OOP')
    - "Calculate Phase noise"
    - "Convert Phase noise to RMS Jitter to be contributed at the PLL output"

Design Contributions

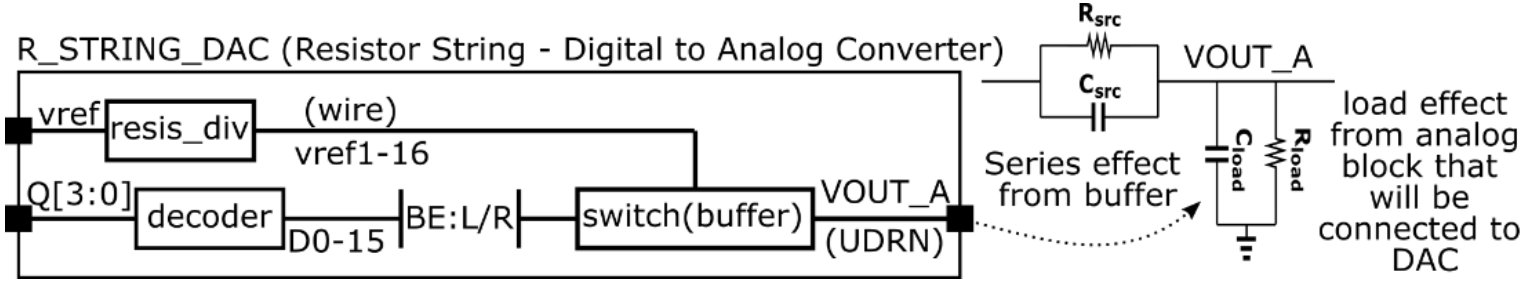
ADC





Fractional – PLL

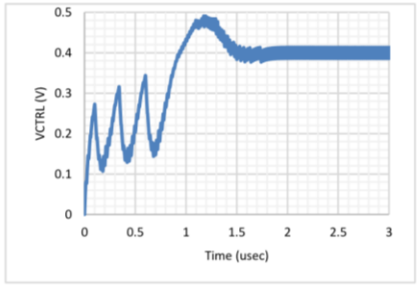
DAC



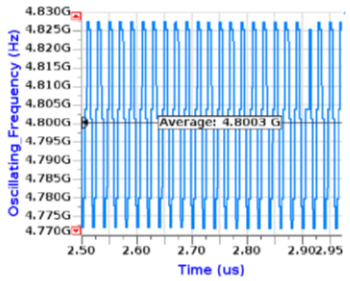
Real Number Modeling – Design Contributions

Outputs that ensure the Accuracy Improvement executed with an event-driven Simulator

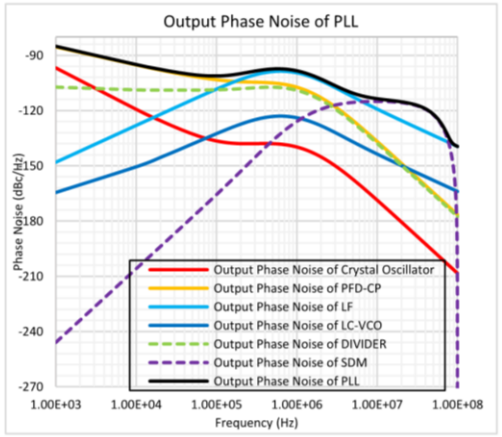
Control Voltage through Simulation time



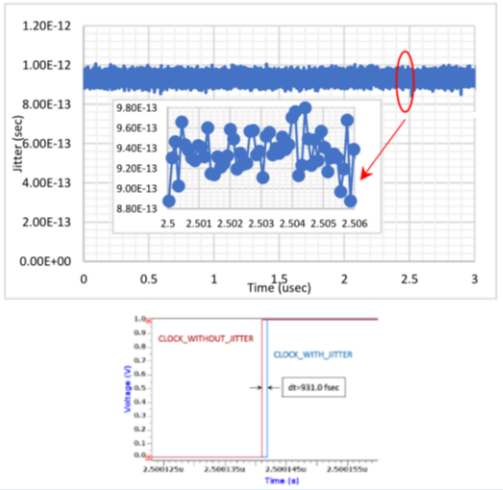
Output Oscillating Frequency through simulation time



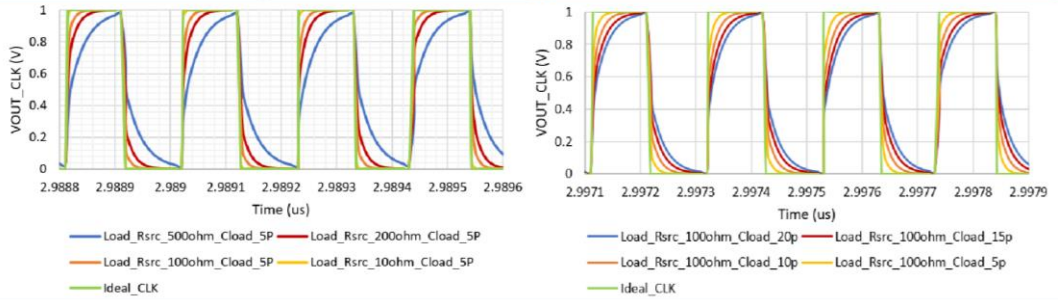
Phase Noise



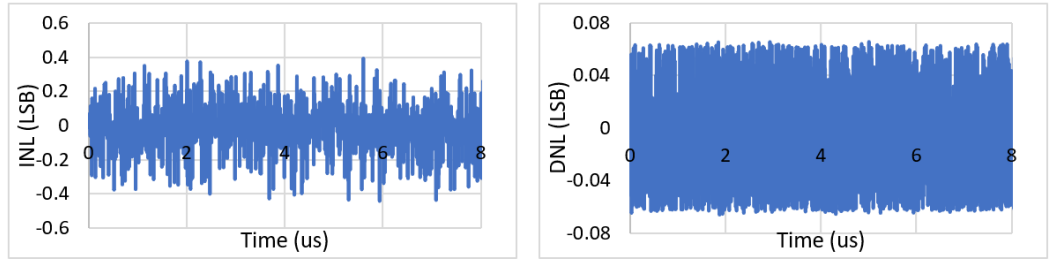
Timing Jitter through simulation time



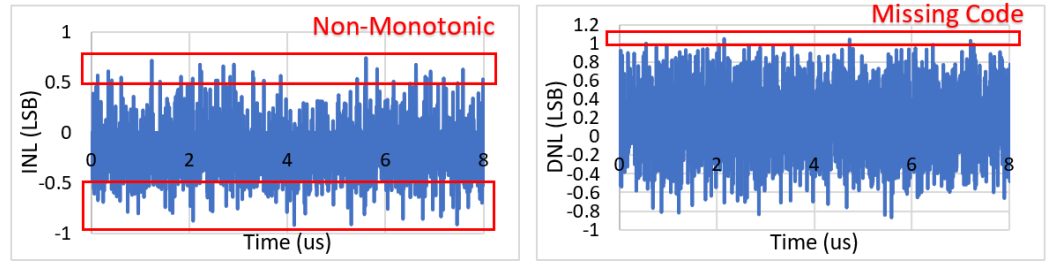
Load Effect



INL/DNL without component mismatch

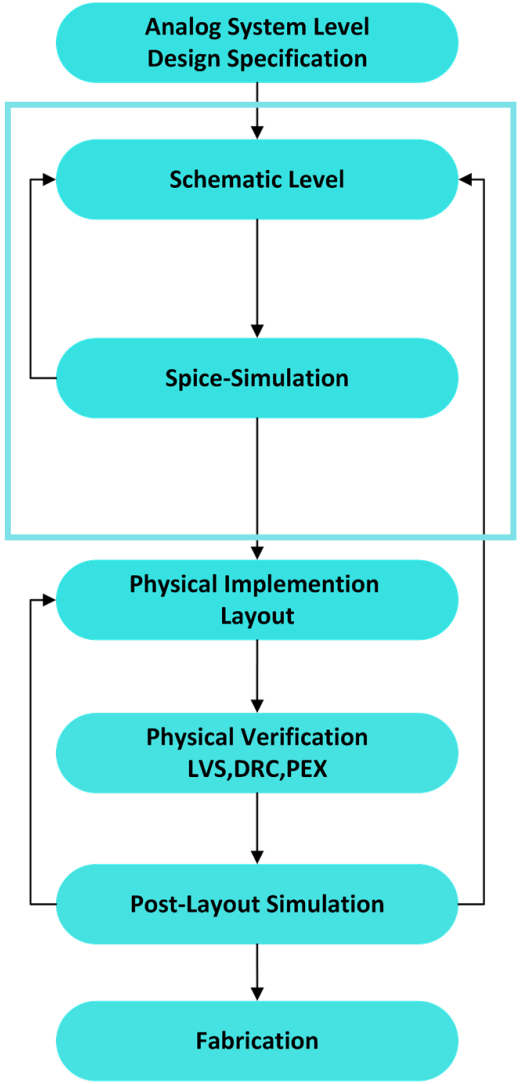


INL/DNL with mismatch component

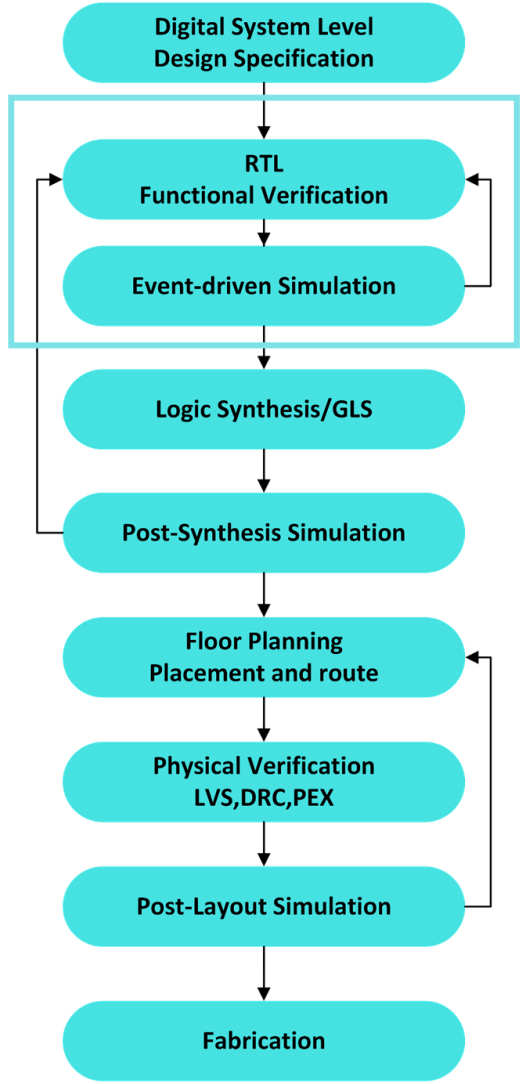


Functional Verification

- Functional verification could start without finishing the analog schematics. There is no need to wait until finishing the analog devices transistor level, to verify the system design.



Analog IC DESIGN



Digital IC DESIGN

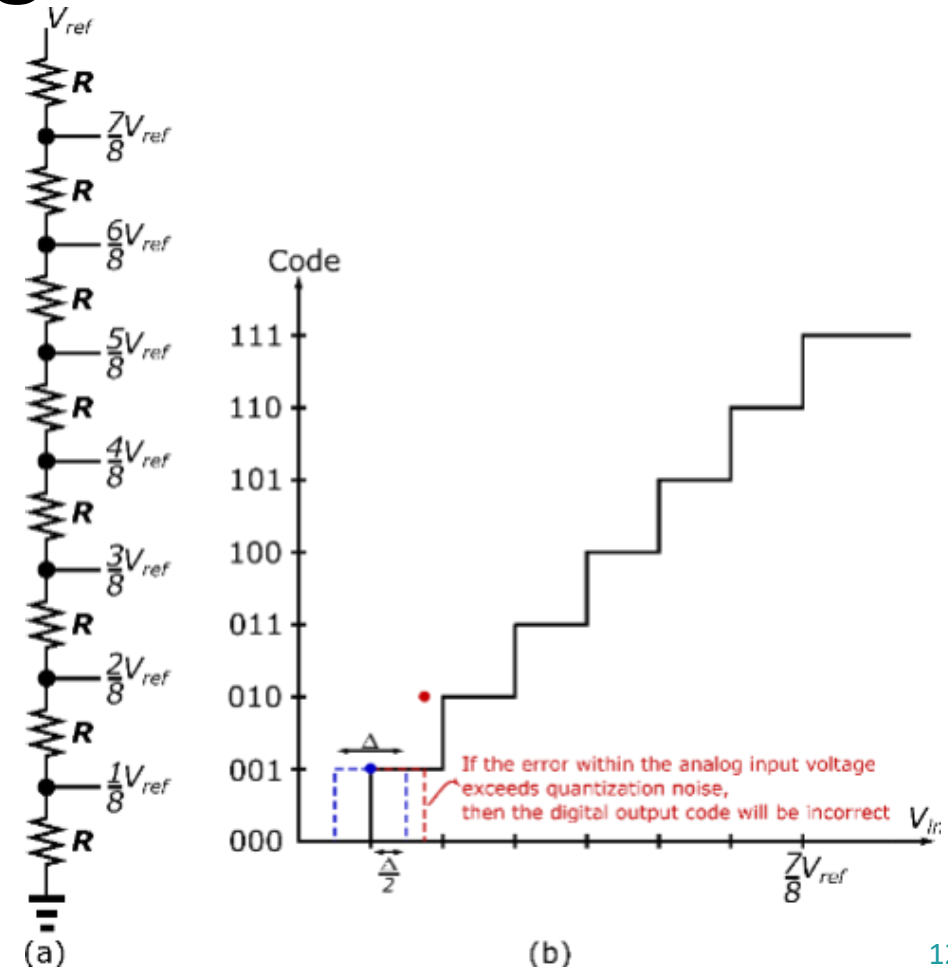
Verification Contributions

During verification of the design, SystemVerilog provides:

- Constrained Random Verification (CRV)
  - Modelling mismatch within ADC quantization noise, and Supply Noise.
- Functional Coverage to cover certain real values
- Assertions/Checkers
  - Relation between I/O ports of ADC/DAC
- UVM based verification, A testbench environment that maintained
  - Interface – Internal signals: Interface to access/record the internal signals of the DUT by binding it to the DUT. This is helpful to force an internal signal by a randomized value.
  - CRV (Sequence Item)
  - Functional Coverage (Subscriber)
  - Assertions/checkers (Scoreboard)

CRV – Randomization as modeling mismatch effect

- ADC/DAC can have a random mismatch in the resistive divider block (a).
- The voltage at each node of the resistive divider will change. The change can be modeled by using the randomization of the reference voltages.
- The constraint is written such that the analog reference voltage will have the quantization noise added or subtracted from it as if the signal exceeds the quantization noise, the output of the ADC will give an error code (b). The worst quantization noise is equal to (Δ/2)
  - where (Δ) is equal to ($\frac{Vref}{2^n}$) and n is the number of levels of conversion



CRV – Writing constraints for the mismatch effect

Writing constraints can be set, in accordance with:

Limits of mismatches within the specified block that are expected from the System-Level engineers to avoid exceeding these limits.

- The outputs are then analyzed to ensure that the block functionality is working correctly according to system level specifications.

Without any limits because the verification engineer was unable to determine the maximum amount of mismatch in the randomized signal.

- However, after debugging, the verification engineer was able to determine the maximum/minimum amount of the mismatch needed to maintain the proper output values.

The verification engineer harden the limit of mismatch.

- Illustrates the potential consequences for the system and the extent to which it may be functionally flawed.

CRV – Randomization of Input Real/logic signals

ADC real Input:

Generate the real input values as a range of the randomized real data and track the output to ensure the functionality of the system is observed.

DAC logic Input:

The input logic signal can be randomized between any value of the start digital code (0) to the end digital code ($2^n - 1$). Or can be randomized sequentially by starting from the first digital code (0) and increasing by the next digital code until reach the last digital code. ($0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \ldots. \rightarrow 2^n - 1$).

Functional Coverage – cover real signal from low to high

The automatic number of the bins is defined according to the ‘real_interval’ that passed as a ‘type_option’ and divided according to the number of real values. Which means, once a bin is created for a value then not another bin is created for this value even if the value is repeated within a different range or declared as a separate value.

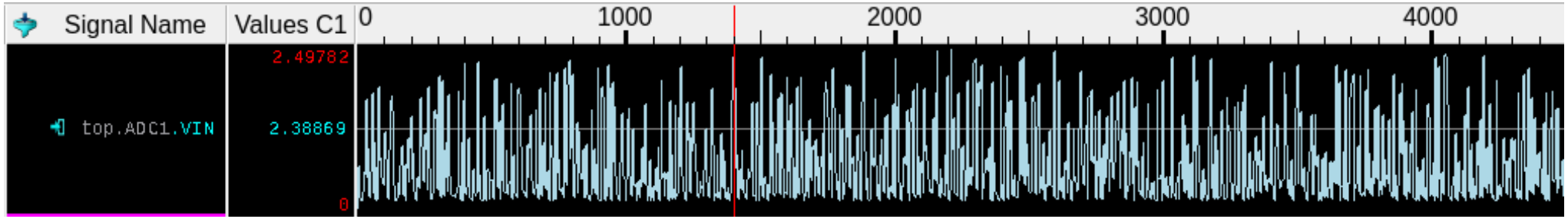
```
// COVERAGE VOLTAGE VALUES WITHIN A CERTAIN RANGE
        real min_a = 0.1; // minimum amplitude
        real max_a = 2.5; // maximum amplitude

        covergroup V_COV @(vout_load);
                option.per_instance = 1;
                coverpoint vout_load {
                        type_option.real_interval = 1E-1;
                        bins v [] = {[min_a:max_a], min_a, max_a};
                }
        endgroup: V_COV
```

type_option.real_interval = x		
bins	values	Coverage status
[min_a : min_a + x]	covers values between min_a to (min_a + x) and the specific value min_a if it's existed	
[min_a + x : min_a + 2x]	covers values between min_a + x to min_a + 2x and the specific value min_a + x if it's existed	
[min_a + 2x : min_a + 3x]	covers values between min_a + 2x to min_a + 3x and the specific value min_a + 2x if it's existed	
…	…	
[max_a – 2x : max_a – x]	covers values between max_a – 2x to max_a – x and the specific value max_a – 2x if it's existed	
[max_a – x : max_a]	covers values between max_a – x to max_a and the specific value max_a – x if it's existed	
[max_a]	covers specific value max_a if it's existed	

VIN	96.00%
vin[0.1:0.2)	1002
vin[0.2:0.3)	592
vin[0.3:0.4)	423
vin[0.4:0.5)	412
vin[0.5:0.6)	218
vin[0.6:0.7)	208
vin[0.7:0.8)	200
vin[0.8:0.9)	156
vin[0.9:1)	182
vin[1:1.1)	106
vin[1.1:1.2)	80
vin[1.2:1.3)	86
vin[1.3:1.4)	76
vin[1.4:1.5)	100
vin[1.5:1.6)	80
vin[1.6:1.7)	98
vin[1.7:1.8)	112
vin[1.8:1.9)	110
vin[1.9:2)	106
vin[2:2.1)	46
vin[2.1:2.2)	34
vin[2.2:2.3)	54
vin[2.3:2.4)	52
vin[2.4:2.5)	38
vin[2.5]	0

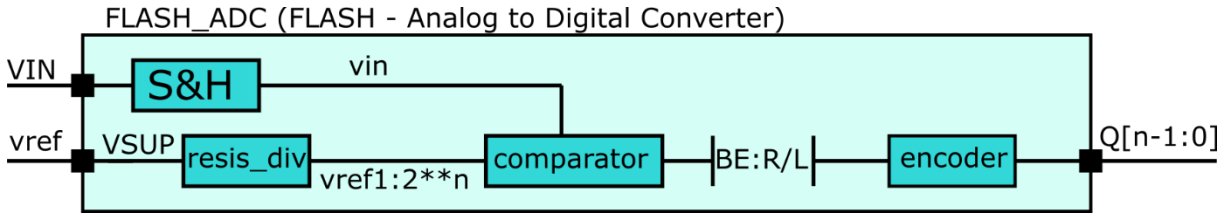
The maximum value of the signal 'vin' is 2.49782. For that the bin 'vin[2.5]' is not hit



Assertions – Relation between I/O ports

ADC:

- Modeled DUT:

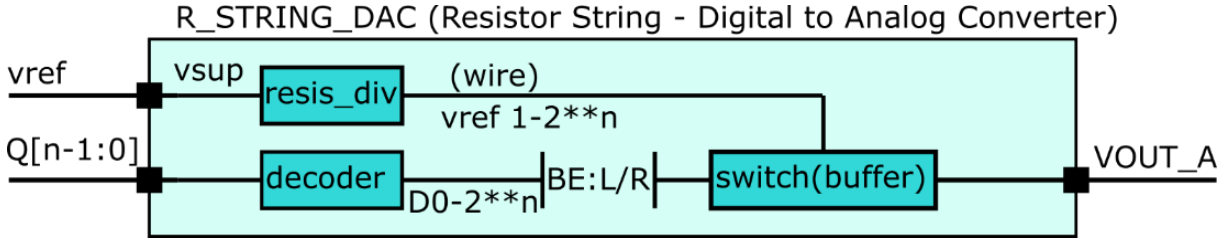


- Relation:

$$logic\ (output) = \left\lfloor \frac{Real\ (input)}{delta} \right\rfloor \ where, delta = \frac{V_{sup}}{2^n}$$

DAC:

- Modeled DUT:

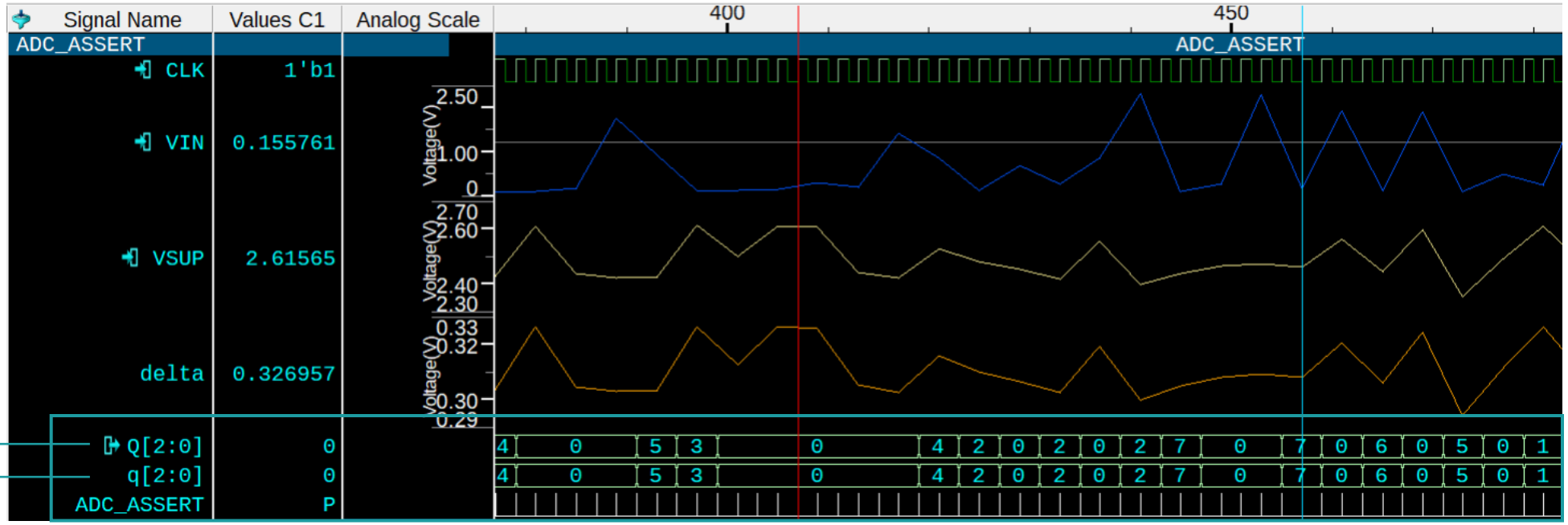


- Relation:

$$Real\ (output)\ =\ Logic\ (input)\ \times\ delta$$

Assertions – comparison

ADC: The assertion (ADC_ASSERT) verifies that the FLASH_ADC is passed (P) when the expected functionality of ADC is observed.

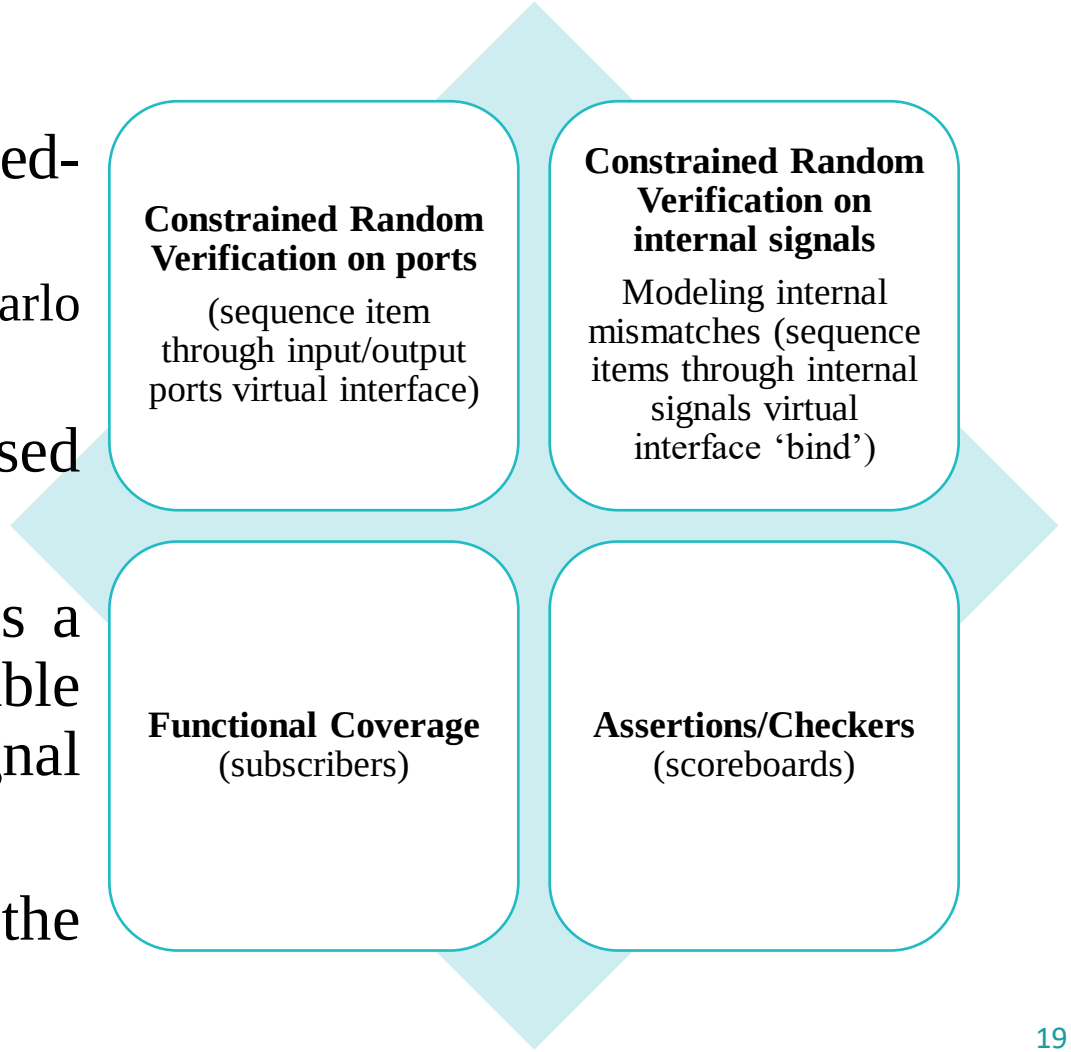


Output from Modeled DUT

Output from Relation

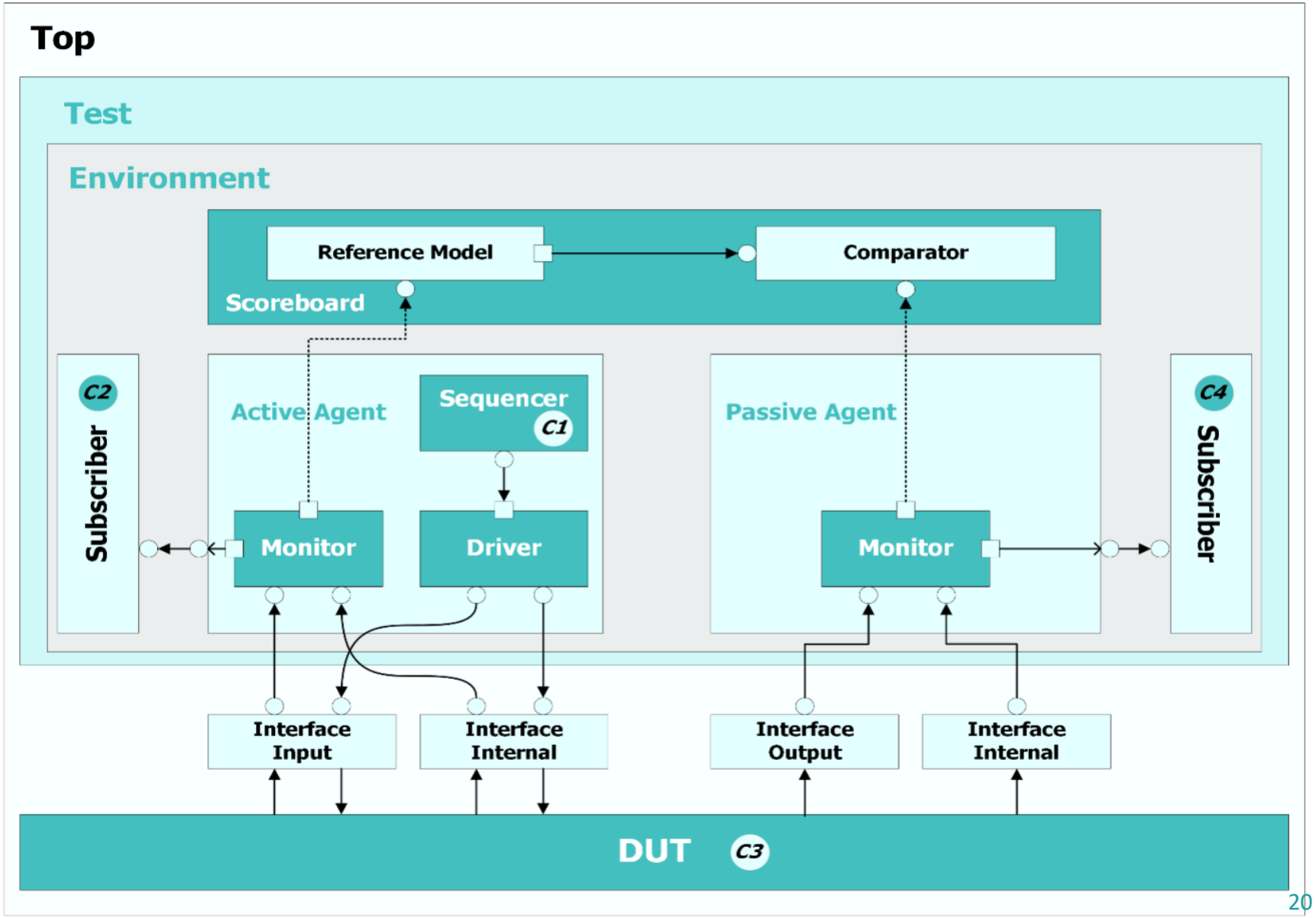
UVM

- The verification of the analog part of the mixed-signal systems, is achieved by hard approaches
  - Direct testing, Corner Analysis, and Monte Carlo simulations
- While the UVM is the most widely used verification standard for the digital circuits.
- As a result, integrating the UVM and RNM is a crucial tactic for creating a fast and reliable verification environment for the mixed-signal devices.
- One environment enables the verification of the analog devices from:



UVM Env.

- Full analog-mixed ADC/DAC modeled DUT is verified using the UVM.
- There are two agents, two subscribers, and a scoreboard in the UVM environment.



UVM – Interfaces

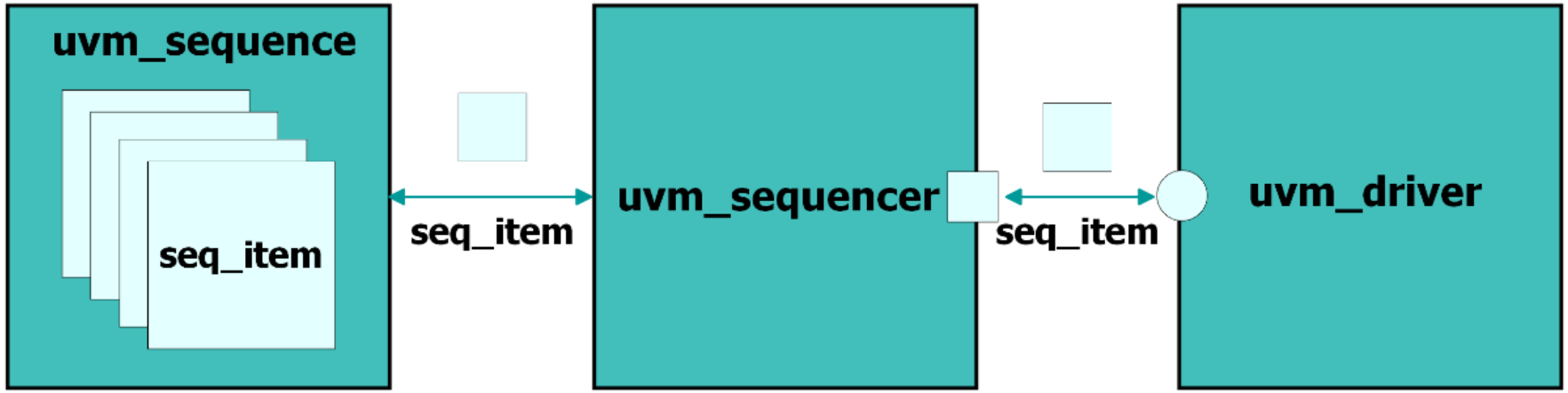
- Interface – Input ports: Interface for accessing the DUT's input ports
- Interface – Output ports: Interface for recording the DUT's output ports
- Interface – Internal signals: Interface for accessing or recording the internal signals of the DUT by binding it to the DUT. This is helpful to force an internal signal by a value or by the randomized value within a constraint random range.

```
import volt_pkg::*; // pkg has a UDN
interface ADC_IF_INT (
  input logic CLK,
  input real vref1, // To access reference voltages
  input real vref2,
  …
  input real vref7,
  output volt D1, // UDN holds resolved voltage
  output volt D2, // To record output of comparator
  …);
endinterface
```

```
module top ();
… // Binding DUT to internal interface
bind FLASH_ADC ADC_IF_INT int_if (.CLK(CLK),
 .vref1(FLASH_ADC.RES_DIV.vref1),
 .vref2(FLASH_ADC.RES_DIV.vref2),
  …
 .D1(FLASH_ADC.COMP.D1),
 .D2(FLASH_ADC.COMP.D2),
  …);
always @(CLK) begin
  force FLASH_ADC.RES_DIV.vref1 = FLASH_ADC.int_if.vref1;
  force FLASH_ADC.RES_DIV.vref2 = FLASH_ADC.int_if.vref2;
  …
end
endmodule
```

UVM – Sequence Generation

The sequence-item, sequence, sequencer, and driver are the four classes needed for the entire sequence generation process.



UVM – Sequence Generation – Input ports

For the ADC device, the reference voltage (VREF) constraint (c_VREF) will have the maximum quantization noise added or subtracted from it as if the signal exceeds the quantization noise, the output of the ADC will give an error code.

The input voltage (VIN) to the ADC will be a randomized input from low voltage (VL) to high voltage (VH).

```
class adc_transaction_in extends  uvm_sequence_item;
      …
// Randomization of Input Signals
   rand real VREF;
   parameter real A_VDD = 2.5;    // Ideal High Supply
   parameter int n = 3;                  // Resolution (Accuracy)
   int n_levels = $pow(2,n);          // Number levels of conversion
   real delta = (A_VDD / n_levels);

   constraint c_VREF {VREF inside {[A_VDD - (delta/2) :  A_VDD + (delta/2)]};};

   rand real VIN;
   real VL = 0.01;
   real VH = 2.5;

   constraint c_VIN  {V_in dist {0.0:/5 , [VL:VH]:/50, [-VH:-VL]:/50};}
      …
endclass: adc_transaction_in
```

UVM – Sequence Generation – Internal signals

The ‘uvm_sequence_item’ class has also the randomization of the internal signals that model the random mismatch within the ADC components.

```
class adc_transaction_in extends uvm_sequence_item;
      …
// Randomization of Internal Signals
 rand real vref1;
 constraint c_vref1 { vref1 inside {[(1*VREF)/n_levels - (delta/2) :  (1*VREF)/n_levels + (delta/2)]}; };

 rand real vref2;
 constraint c_vref2 { vref2 inside {[(2*VREF)/n_levels - (delta/2) : (2*VREF)/n_levels + (delta/2)]}; };
 …
endmodule
```

UVM – Monitors

The environment here has two UVM monitors:

The first one captures:

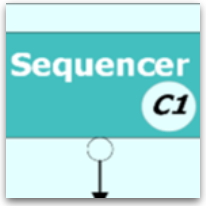
- The input signals to the DUT through the virtual interface responsible for accessing the DUT's input ports.
- Moreover, the internal signals of the DUT through the virtual interface responsible for accessing the DUT's internal signals that the verification engineer needs to force their values.
- This monitor is inside an active agent.

The second one captures:

- The output signals to the DUT through the virtual interface responsible for recording the DUT's output ports.
- Moreover, the internal signals of the DUT through the virtual interface responsible for recording the DUT's internal signals that the verification engineer needs to ensure their values.
- This monitor is inside a passive agent.

UVM – Subscribers

Functional cover points are placed in four places:

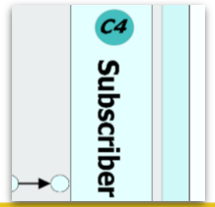


Close to the randomization to guarantee that the system is set to all expected randomized values within a given range.
As a result, the sequence declares the cover points. *(C1)*

Close to the system's input to make sure that all expected values are present at the DUT's input ports.
'uvm_subscriber', which was designed to handle data that are tracked from the input interface, can accomplish this. *(C2)*

Located within the DUT to guarantee that the internal signals contain the right values. *(C3)*

Close to the system's output ports to guarantee that it covers all anticipated output values.
The verification engineer then ensures that the system's output values are being observed from this functional cover point.
'uvm_subscriber', which is designed to cover the data monitored from the output interface, can accomplish this. *(C4)*



UVM – Scoreboard

Reference model is a simple representation to the functionality of the DUT that links the expected output and the input received from the DUT with a simple relation.

ADC:

$$logic\ (output) = \left\lfloor \frac{Real\ (input)}{delta} \right\rfloor\ where, delta = \frac{V_{sup}}{2^n}$$

DAC:

$$Real\ (output)\ =\ Logic\ (input)\ \times\ delta$$

The output transaction from the reference model is then compared with the output transaction received by the DUT.

UVM – Scoreboard – Code

ADC

```
class refmod extends uvm_component;
    `uvm_component_utils(refmod);
    …
    int n = 3;
    int nlevels = 2**n; //No_levels_of_conversion

virtual task run_phase(uvm_phase phase);
 super.run_phase(phase);
 forever begin
  in.get(tr_in);
   if (tr_in.VIN >= vsuplow && VIN < ((tr_in.VREF)/n_levels)) begin
    tr_out.Q = `0;
   end

   else if (tr_in.VIN >= ((n_levels-1) && tr_in.VIN <= tr_in.VREF)
   begin
    tr_out.Q = `1;
   end

   else if ((tr_in.VIN >= ((tr_in.VREF)/n_levels))
          && (tr_in.VIN < ((n_levels-1) * ((tr_in.VREF)/n_levels))))
   begin
    tr_out.Q = $floor(tr_in.VIN ((tr_in.VREF)/n_levels));
   end
 out.put(tr_out);
 end
endtask: run_phase

endclass: refmod
```

DAC

```
class refmod extends uvm_component;
    `uvm_component_utils(refmod);
    …
    int n = 3;
    int nlevels = 2**n; //No_levels_of_conversion

virtual task run_phase(uvm_phase phase);
 super.run_phase(phase);
 forever begin
  in.get(tr_in);
   if (tr_in.Q > '0 && tr_in.Q < `1) begin
    tr_out.vout_a = Q *((tr_in.VREF)/nlevels);
   end

   else if (tr_in.Q == '1)
   begin
    tr_out.vout_a = ((nlevels-1) * ((tr_in.VREF)/nlevels));
   end

   else if (tr_in.Q == '0)
   begin
    tr_out.vout_a = 0;
   end
  out.put(tr_out);
 end
endtask: run_phase


endclass: refmod
```

Check on ADC/DAC Linearity

- The functional correctness of the converter is measured by the quantization error.
- The quantization error is the difference between the infinite resolution and the actual characteristics. It's equal to $\frac{1}{2}\ LSB = \ \pm\frac{\Delta}{2} = \frac{V_{FS}}{2^{n+1}}$ and considered as a noise added to the signal.
- The converter errors should be less than the quantization noise. The circuit errors are due to:
  - The Component random mismatch due to the fabrication tolerances.
  - The limitation in build block specification like gain, bandwidth, linearity, …

Check on Linearity – INL/DNL

Checking the quantization errors effect in the converter functionality, can be derived by measuring the ADC linearity. Which means measuring the transition deviations of the converter from the ideal characteristics.

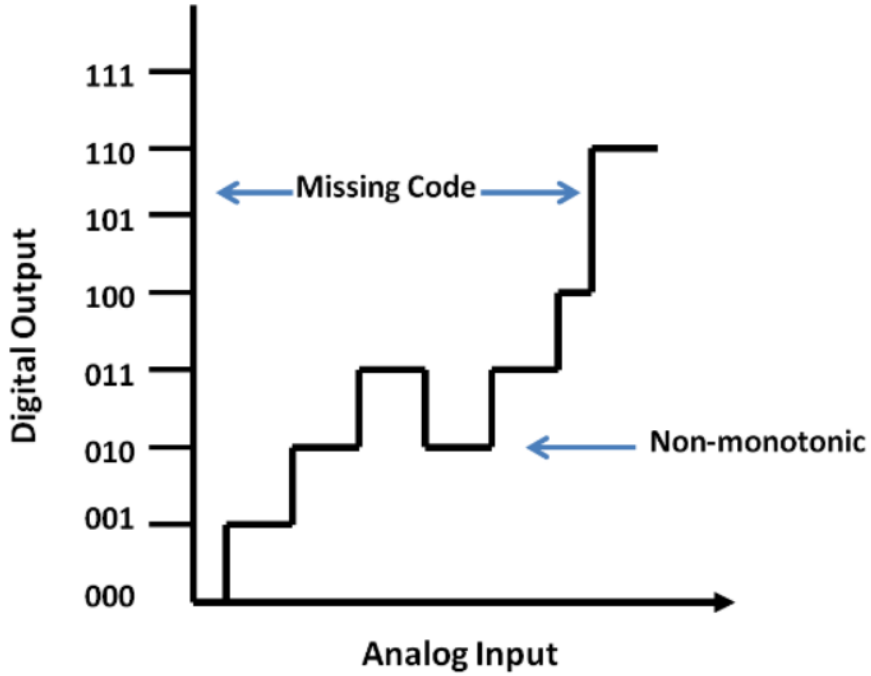
This is known by finding the *Integral Non-Linearity* (INL) and *Differential Non-Linearity* (DNL).

- INL: Maximum deviation of code transitions from their ideal values in LSB. A converter is guaranteed to be monotonic if the maximum INL is less than ± 0.5 LSB.

$$INL\ (in\ LSB) = \frac{V_{i\_real} - V_{i\_ideal}}{\Delta}$$

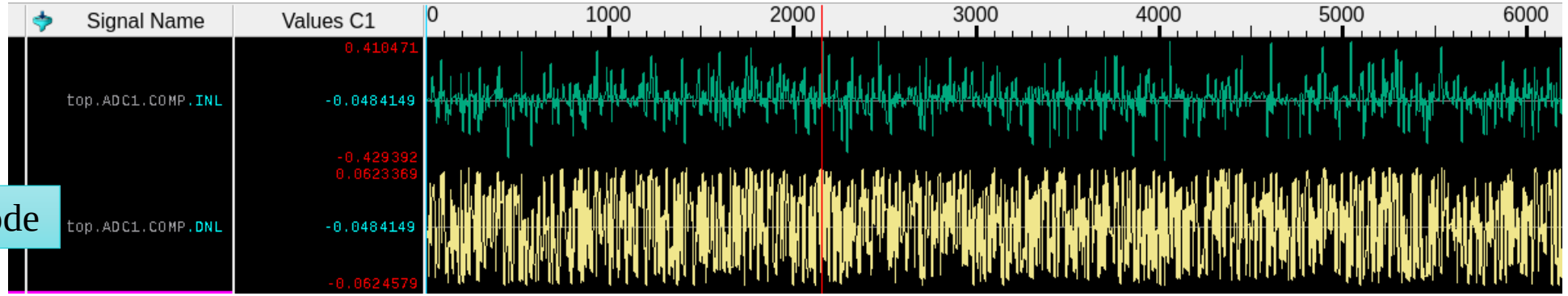
- DNL: Maximum deviation in the step width from the ideal values of Δ in LSB. If $|DNL| \geq 1\ LSB$, this will result in a missing code.

$$DNL\ (in\ LSB) = \frac{V_i - V_{i-1}}{\Delta} - 1$$



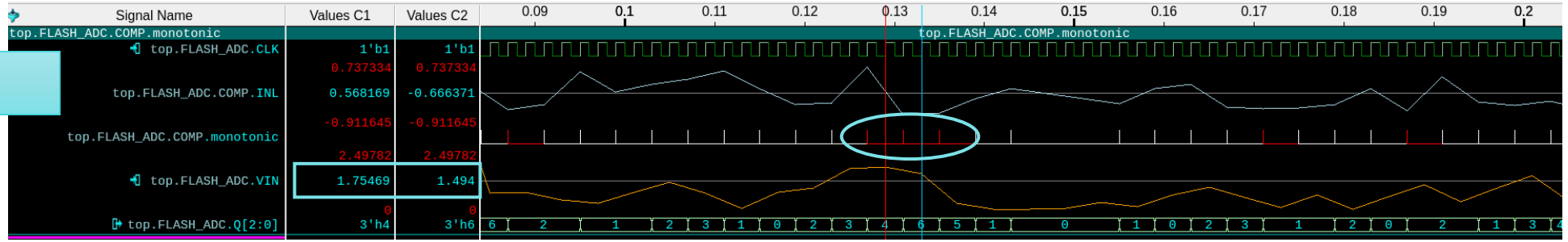
INL/DNL

Monotonic/Non-Missing Code



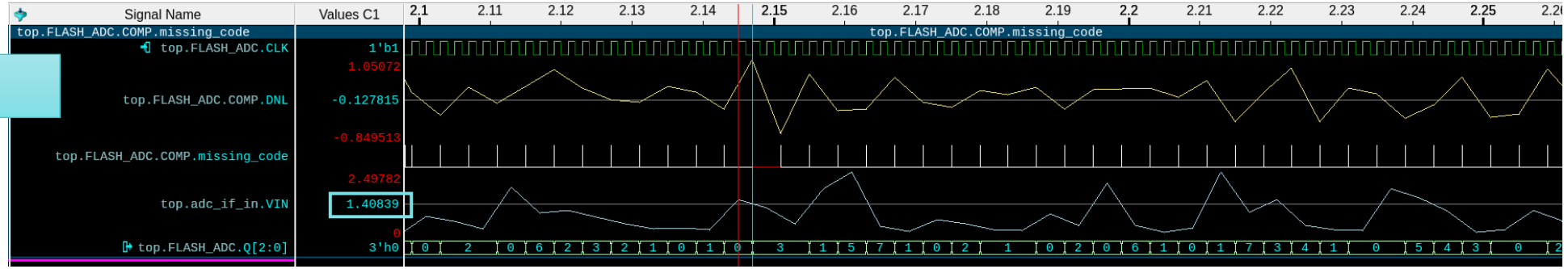
INL and DNL values with acceptable modeling component mismatch.

Non monotonic behavior



Apply extent quantization noise on reference voltage and voltage nodes of resistive divider. Converter could be non-monotonic

Missing Code



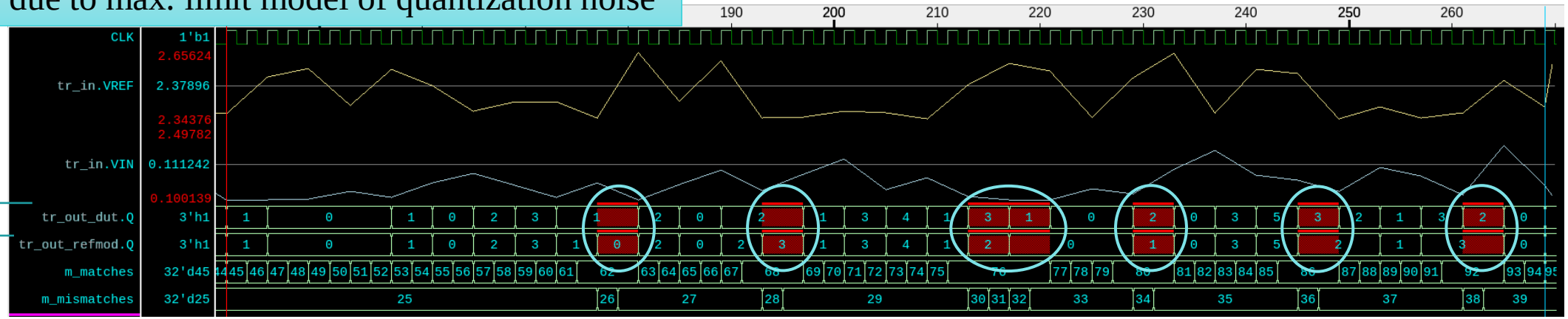
Apply extent quantization noise on reference voltage and voltage node of resistive divider. Converter could have missing-code

Results - Scoreboard

Scoreboard Outputs due to max. limit model of quantization noise

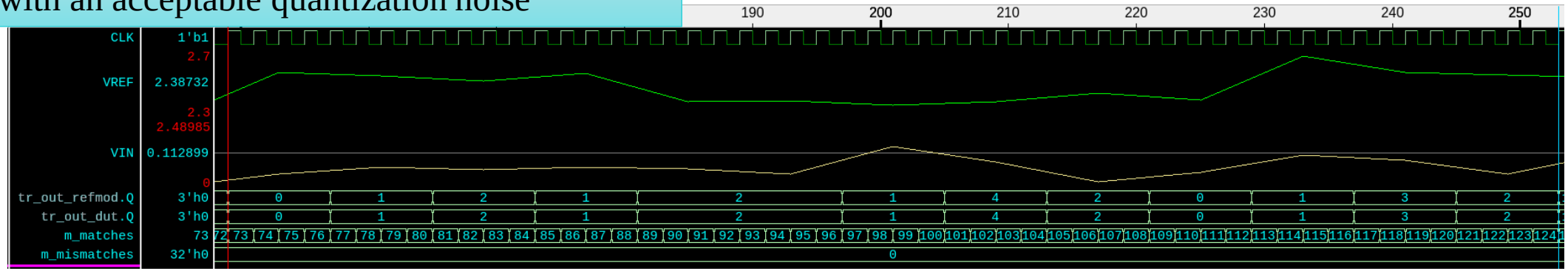
DUT Output

Reference Model Output



m_mismatches, mismatch between output of DUT and reference model, increase with simulation time

Scoreboard Outputs with an acceptable quantization noise



m_mismatches, mismatch between output of DUT and reference model, equal to 0

Summary

In the digital environment, there are a lot of verification techniques that can be used to find bugs within a system. This makes the digital verification is always preferred as its reliability, and usage. Only one environment and one event-driven simulator can provide these verification techniques in an automated way. The paper illustrates:

- The constrained random verification. Such as randomizing the input components of ADC/DAC as modeling quantization mismatch in ADC/DAC. From this step, the digital verification engineer will be able to know the extent of the variation that each component can hold else the system will behave incorrectly.
- The functional coverage ensures that the electrical voltage is covered under a certain amplitude range.
- Assertions are built to check the functionality of the whole system if there is a simple relation between the output and input of a system.
- The UVM-based verification is one testbench environment that provides classes to support the randomization of ports through ‘uvm_sequence’ class. Supports the functional coverage through ‘uvm_subscriber’ class and assertions through ‘uvm_scoreboard’ class.

Questions

Thanks for listening. Appreciated.

Do you have any Question to ask ? ☺ …

Contact: mariam.maurice@siemens.com

2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Thank You

accellera
SYSTEMS INITIATIVE