2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

Addressing Advanced Mixed-Signal Verification Scenarios by Developing a UVM Framework for Analog Models

Simul Barua - Ulkasemi, Inc.
Shahriar Kabir - Ulkasemi, Inc.
Henry Chang - Designer's Guide Consulting, Inc.

accellera SYSTEMS INITIATIVE

Agenda

- Motivation and purpose of the AMSV utility framework
- Structure of the framework
- Case study: application of the framework
- Framework implementation in models and testbench
- Simulation results
- Summary & future scopes

Increased Complexity in IC Verification

- Modern applications outpaces design complexity
- Declining verification success rates
- Key factors include analog design issues and mixed-signal interface bugs.
- Insights from our previous research
  - Integrating validated functional analog models enhances chip-level verification.
  - Improved bug detection at the system level
- Remaining gap
  - Traditional methodologies still lack the robustness needed to address more complex system requirements

Evolving Verification Needs for the Digital Twin

- Must include analog elements
- System performance
  - Simulating temperature effects, parametric variations, and analog non-idealities
  - Ensuring real-world performance criteria are met
- Safety Requirements
  - Modeling how analog blocks, subsystems, and systems can fail
  - Verifying fault tolerance and mitigation strategies
- Conventional verification methodologies lack the support needed for these critical features

Limitations of Traditional Methodologies

- Transistor-level simulations for real-world complex scenarios
  - Accurate, yet impractical for system-level verification
  - Environmental effects (e.g. temperature) demand prolonged runs
- Verifying critical features using system-level testbenches
  - No direct control over environmental factors (e.g. temperature)
  - Difficult to observe system responses (e.g. thermal shutdowns)
  - Predictable failures require design modifications accommodating fault conditions
- Need for a flexible and efficient approach
  - Designs must accommodate the verification of both environmental and safety features
  - System-level verification must efficiently handle complex, mixed-signal scenarios efficiently
  - Faster simulation time

Overcoming the Limitations

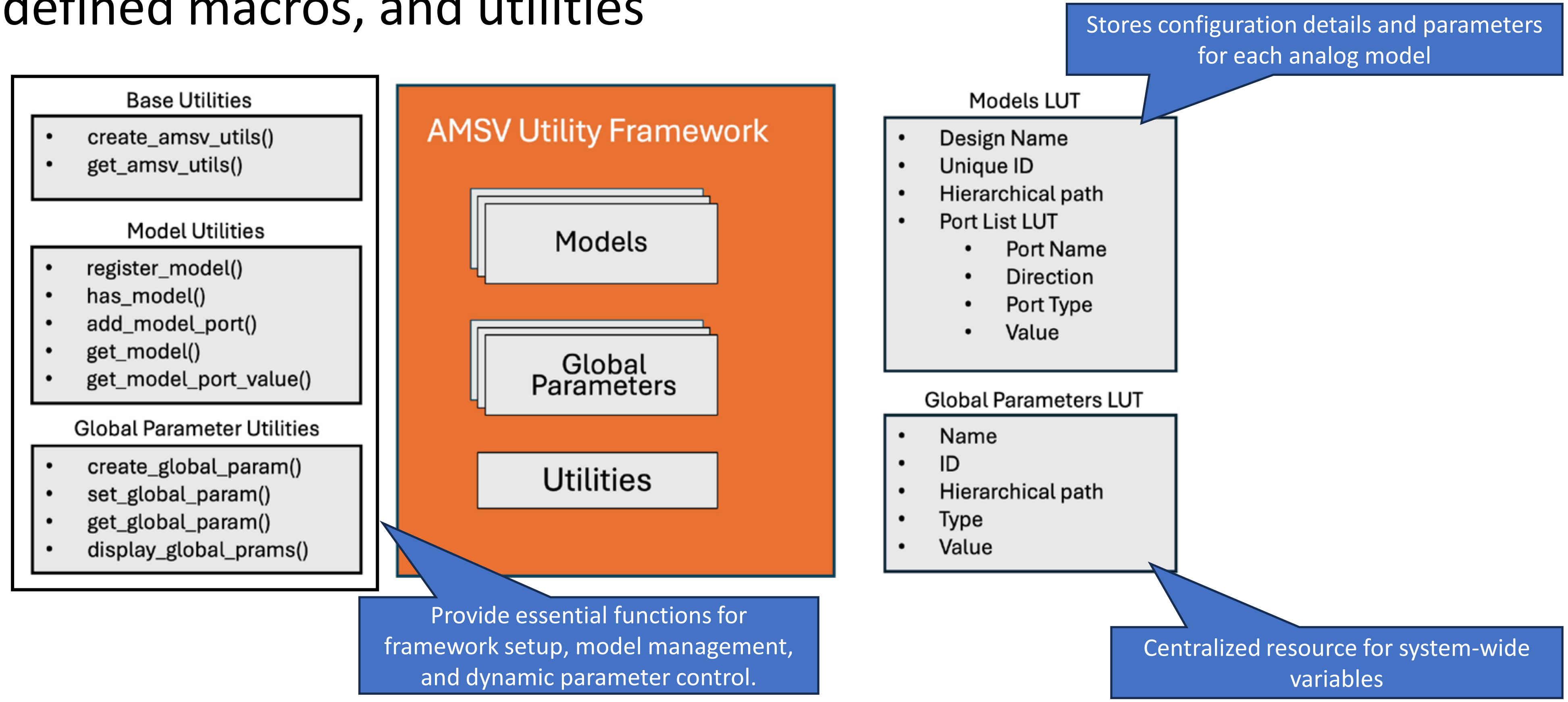
- Leveraging analog models
  - Feasible simulation speeds compared to transistor-level
  - Straightforward incorporation of complex effects (e.g. environmental stress, parametric shifts)
  - Enables real-world scenario modeling (e.g. AI performance under analog degradation)
- Enhancing the UVM Testbench
  - Enhanced monitoring and control for analog models
- Bridging the gap
  - A dedicated mechanism is required to enhance both the analog model and the UVM testbench

The AMSV Utility Framework

- Overview
  - Leverages SystemVerilog OOP constructs
  - Enhances analog models and system-level UVM testbench
  - Efficient communication between the models and the testbench
  - Simulates real-world conditions: temperature effects, parametric variations, and failures
- Addressing limitations of traditional methodologies
  - Enables modeling of real-world scenarios
  - Faster, practical alternative to slow transistor-level simulations
  - Enables direct control and monitoring of environmental impacts
- Key features
  - Introduces bidirectional data flow for advanced observation and control without compromising pin accuracy
  - Dynamic parameter adjustment from UVM testbench
  - Continuous fault detection and debugging via model-to-testbench reporting

Structure of the AMSV Framework

- SystemVerilog package with OOP classes, LUTs, pre-defined macros, and utilities

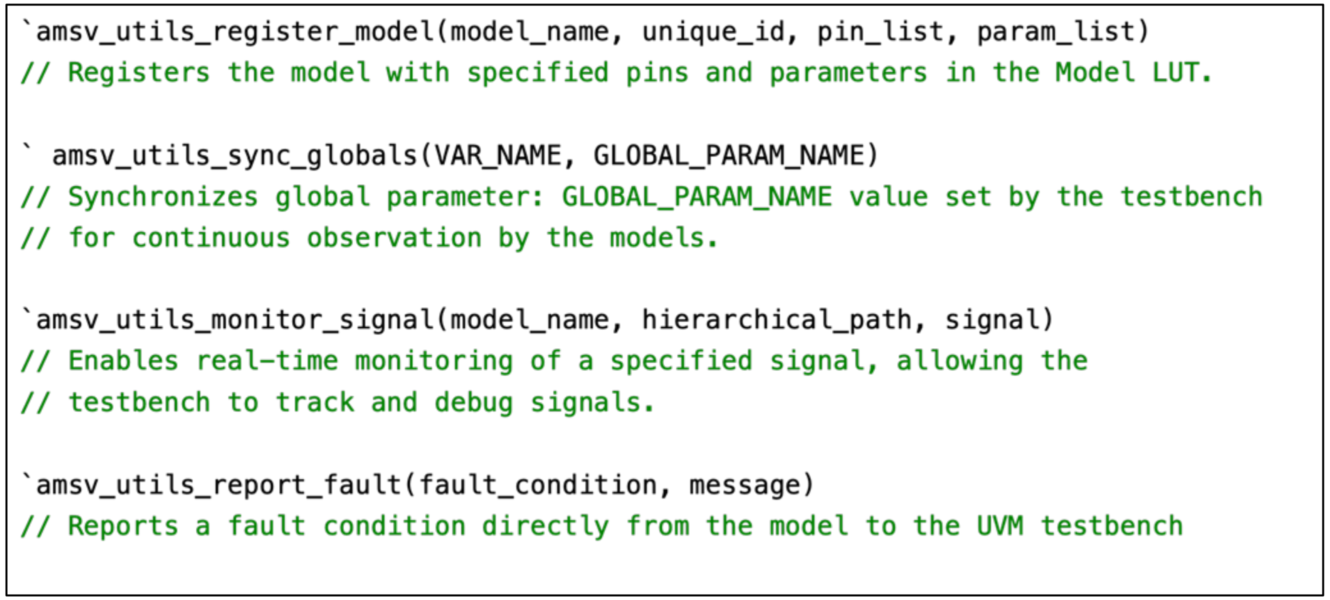


Code Snippet of the AMSV Utility Package

```
package amsv_utils_pkg;
  // Model LUT
  class model_utilities;
  // Global Paremeter LUT
  class global_utilities;
  // Other Helper Signals of the package
  ............................
  // The Base Class of the AMSV Utility Framework
  class amsv_utilities;
    // AMSV utility singleton class
    static amsv_utilities amsv_utils = null;
    // Lookup table consisting model details
    model_utilities models[string];
    // Lookup table for global variables
    global_utilities globals[string];
    ............................
    // Utilities
    // Private constructor to prevent direct instantiation
    extern protected function new(string name);
    // Create instance of AMSV Utility
    extern static function amsv_utilities create_amsv_utils(string name);
    // Get instance of AMSV Utility
    extern static function amsv_utilities get_amsv_utils();
    ............................
    // Model LUT Utilities
    // Registers Analog model in lookup table with name, unique ID and hierarachy
    extern function void register_model(string name = "", string id = "", string hier = "");
    // Returns 1 when model exists with provided unique ID
    extern function bit has_model(string id);
    // Adds port information of registered model
    extern function void add_model_port(string id, string name, string dir, string port_type);
    // Returns model utility based on unique ID
    extern function model_utilities get_model(string id);
    // Returns the port description of the model based on unique model ID and port name
    extern function port_t get_model_port(string id, string port_name);
    ............................
    // Global Parameter Utilities
    // Creates global variable with name, data type and value
    extern function void create_global_param(string name = "", string data_type = "", real value);
    // Sets global value during runtime
    extern function void set_global_param(string name, real value);
    // Returns global value with provided name
    extern function real get_global_param(string name);
    // Returns 1 when global value with provided name exists
    extern function bit has_global_param(string name);
    ............................
    // Display Utilities
    ............................
  endclass
endpackage
```

Pre-defined Utility Macros

- Simplify integration and reduce repetitive code
- Ensure consistent usage across models and verification environments
- Minimize complexity, making the framework more accessible and efficient
- Key Macros
    - Model registration macros
    - Global parameter macros
    - Signal monitoring macros
    - Fault reporting macros



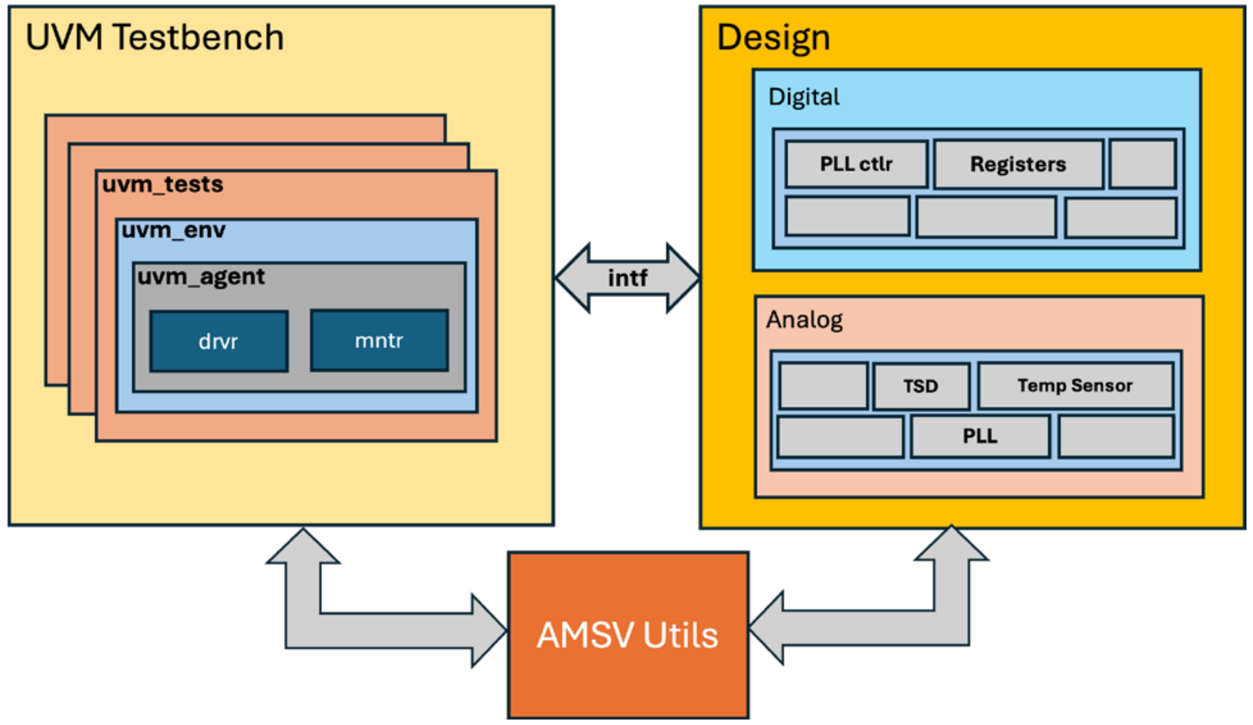
```
`amsv_utils_register_model(model_name, unique_id, pin_list, param_list)
// Registers the model with specified pins and parameters in the Model LUT.

` amsv_utils_sync_globals(VAR_NAME, GLOBAL_PARAM_NAME)
// Synchronizes global parameter: GLOBAL_PARAM_NAME value set by the testbench
// for continuous observation by the models.

`amsv_utils_monitor_signal(model_name, hierarchical_path, signal)
// Enables real-time monitoring of a specified signal, allowing the
// testbench to track and debug signals.

`amsv_utils_report_fault(fault_condition, message)
// Reports a fault condition directly from the model to the UVM testbench
```

Case Study: Application of the AMSV Framework

- Design Overview
  - System-on-chip (SoC) design with digital and analog components.
  - Includes a Power Management Unit (PMU) and Phase-Locked Loop (PLL)
- Key Objectives
  - Integrate AMSV framework with analog models and UVM testbench
  - Verify complex scenarios like environmental impacts and safety mechanisms
- Critical analog components
  - PLL: sensitive to system temperature, leading to frequency drift; monitored by a digital control unit for compensation
  - Thermal shutdown (TSD): monitors system temperature; designed to shut down upon reaching unsafe limits

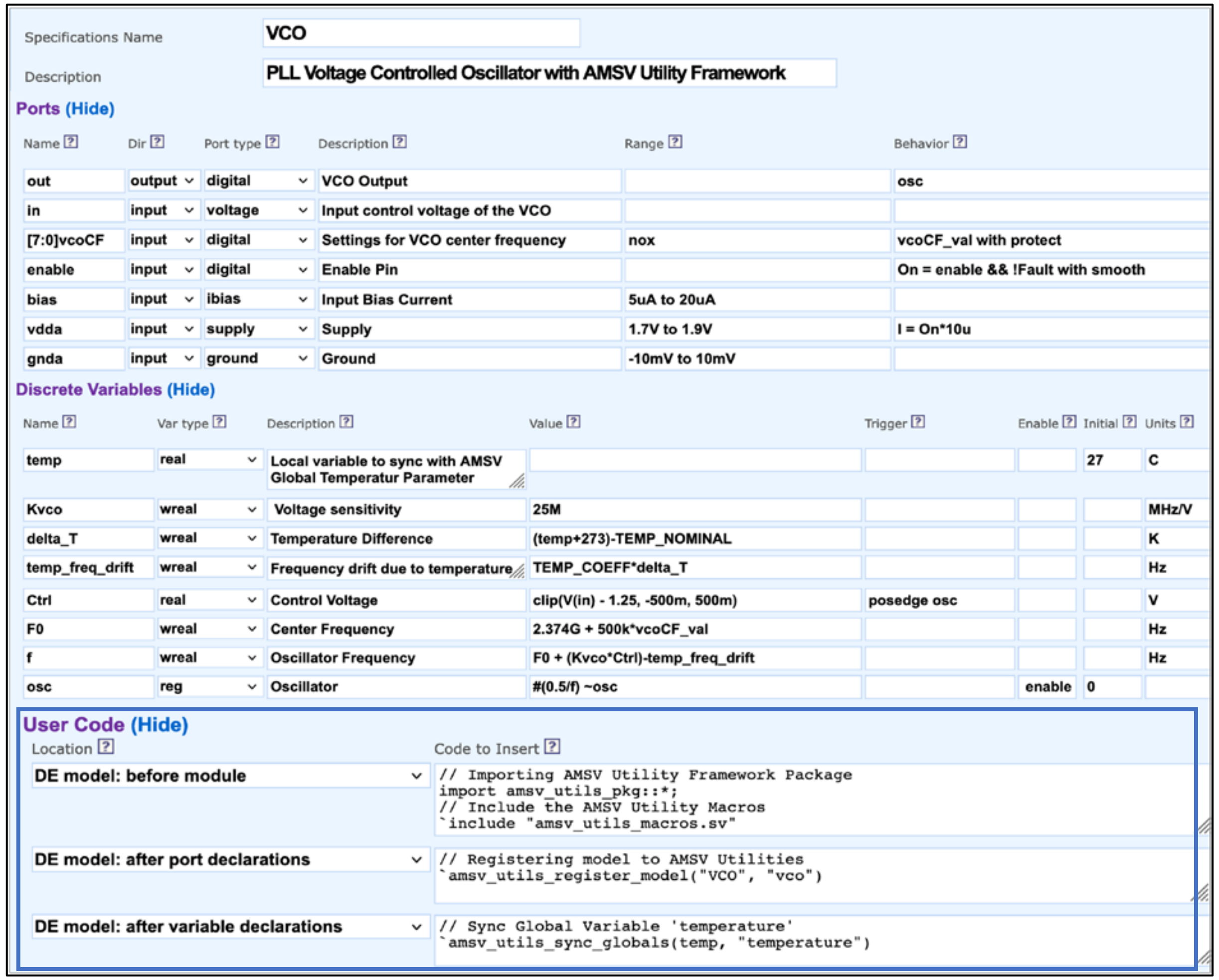
Case Study: AMSV Integration and Verification Scenarios

- Verification scenarios
  - Verify temperature effects on PLL frequency drift
  - Assess system response when the TSD block fails to trigger
- AMSV framework integration
  - Configure parameters and inject faults via UVM testbench
  - Real-time control and monitoring through the AMSV framework



Modeling Analog Blocks & AMSV Integration

- Used SystemVerilog discrete electrical (user-defined nettype) approach
- Generated validated models and self-checking MVS testbenches with the Model-in-Minutes (MiM) tool
    - Automatically generates models from a specification
    - Allows user-defined custom code integration
    - Enabled seamless AMSV framework integration across multiple models



Specifications Name	VCO
Description	PLL Voltage Controlled Oscillator with AMSV Utility Framework

Ports (Hide)

Name	Dir	Port type	Description	Range	Behavior
out	output	digital	VCO Output		osc
in	input	voltage	Input control voltage of the VCO		
[7:0]vcoCF	input	digital	Settings for VCO center frequency	nox	vcoCF_val with protect
enable	input	digital	Enable Pin		On = enable && !Fault with smooth
bias	input	ibias	Input Bias Current	5uA to 20uA	
vdda	input	supply	Supply	1.7V to 1.9V	I = On*10u
gnda	input	ground	Ground	-10mV to 10mV	

Discrete Variables (Hide)

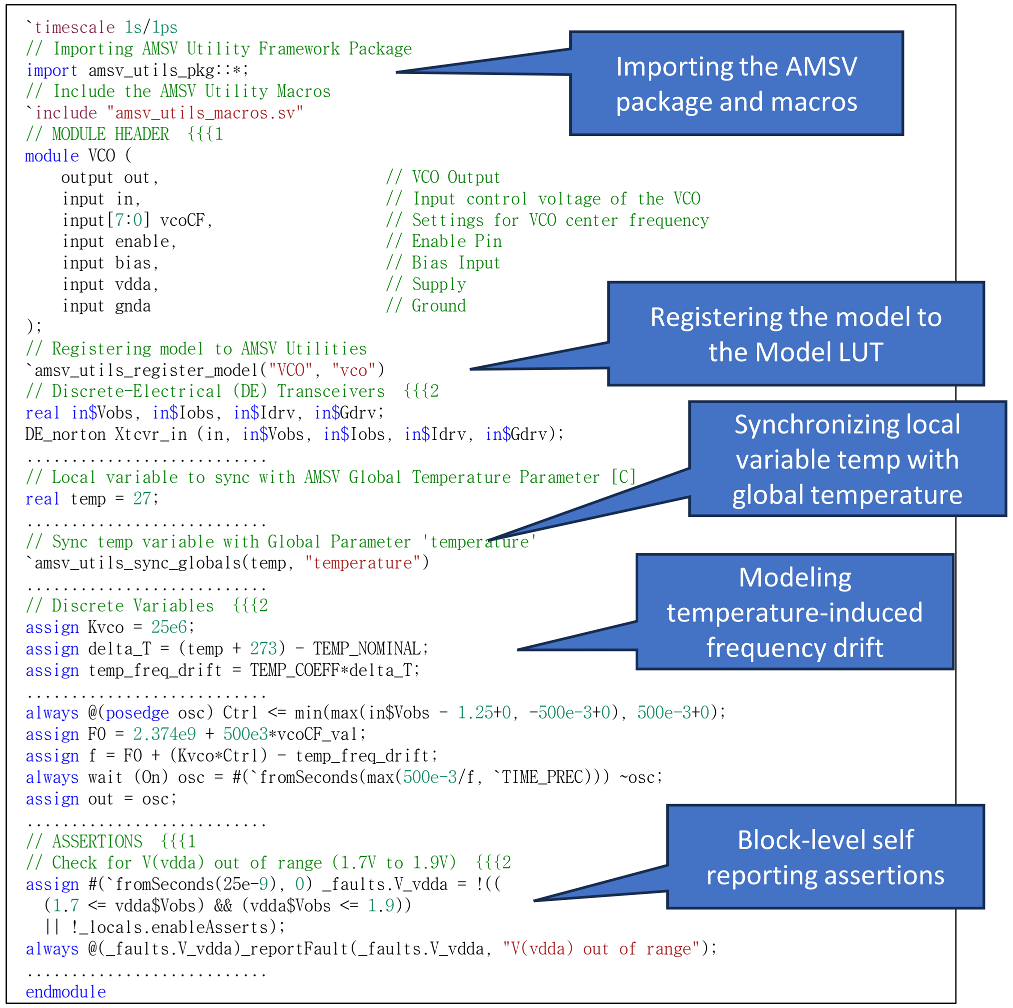
Name	Var type	Description	Value	Trigger	Enable	Initial	Units
temp	real	Local variable to sync with AMSV Global Temperatur Parameter				27	C
Kvco	wreal	Voltage sensitivity	25M				MHz/V
delta_T	wreal	Temperature Difference	(temp+273)-TEMP_NOMINAL				K
temp_freq_drift	wreal	Frequency drift due to temperature	TEMP_COEFF*delta_T				Hz
Ctrl	real	Control Voltage	clip(V(in) - 1.25, -500m, 500m)	posedge osc			V
F0	wreal	Center Frequency	2.374G + 500k*vcoCF_val				Hz
f	wreal	Oscillator Frequency	F0 + (Kvco*Ctrl)-temp_freq_drift				Hz
osc	reg	Oscillator	#(0.5/f) ~osc		enable	0	

User Code (Hide)

Location	Code to Insert
DE model: before module	`// Importing AMSV Utility Framework Package` `import amsv_utils_pkg::*;` `// Include the AMSV Utility Macros` ``` `include "amsv_utils_macros.sv" ```
DE model: after port declarations	`// Registering model to AMSV Utilities` ``` `amsv_utils_register_model("VCO", "vco") ```
DE model: after variable declarations	`// Sync Global Variable 'temperature'` ``` `amsv_utils_sync_globals(temp, "temperature") ```

Integrating the AMSV Framework into Models

- *`amsv_utils_register_model* macro registers the model in Model LUT
- *`amsv_utils_sync_global* macro synchronizes model variable *temp* with *Global LUT* parameter *temperature*
- Models frequency drift due to global temperature changes
- AMSV framework reports block-level assertions directly to the system-level testbench



```
`timescale 1s/1ps
// Importing AMSV Utility Framework Package
import amsv_utils_pkg::*;
// Include the AMSV Utility Macros
`include "amsv_utils_macros.sv"
// MODULE HEADER  {{{1
module VCO (
    output out,                          // VCO Output
    input in,                            // Input control voltage of the VCO
    input[7:0] vcoCF,                    // Settings for VCO center frequency
    input enable,                        // Enable Pin
    input bias,                          // Bias Input
    input vdda,                          // Supply
    input gnda                           // Ground
);
// Registering model to AMSV Utilities
`amsv_utils_register_model("VCO", "vco")
// Discrete-Electrical (DE) Transceivers  {{{2
real in$Vobs, in$Iobs, in$Idrv, in$Gdrv;
DE_norton Xtcvr_in (in, in$Vobs, in$Iobs, in$Idrv, in$Gdrv);
...........................
// Local variable to sync with AMSV Global Temperature Parameter [C]
real temp = 27;
...........................
// Sync temp variable with Global Parameter 'temperature'
`amsv_utils_sync_globals(temp, "temperature")
...........................
// Discrete Variables  {{{2
assign Kvco = 25e6;
assign delta_T = (temp + 273) - TEMP_NOMINAL;
assign temp_freq_drift = TEMP_COEFF*delta_T;
...........................
always @(posedge osc) Ctrl <= min(max(in$Vobs - 1.25+0, -500e-3+0), 500e-3+0);
assign F0 = 2.374e9 + 500e3*vcoCF_val;
assign f = F0 + (Kvco*Ctrl) - temp_freq_drift;
always wait (On) osc = #(`fromSeconds(max(500e-3/f, `TIME_PREC))) ~osc;
assign out = osc;
...........................
// ASSERTIONS  {{{1
// Check for V(vdda) out of range (1.7V to 1.9V)  {{{2
assign #(`fromSeconds(25e-9), 0) _faults.V_vdda = !((
  (1.7 <= vdda$Vobs) && (vdda$Vobs <= 1.9))
  || !_locals.enableAsserts);
always @(_faults.V_vdda)_reportFault(_faults.V_vdda, "V(vdda) out of range");
...........................
endmodule
```

Importing the AMSV package and macros
Registering the model to the Model LUT
Synchronizing local variable temp with global temperature
Modeling temperature-induced frequency drift
Block-level self reporting assertions

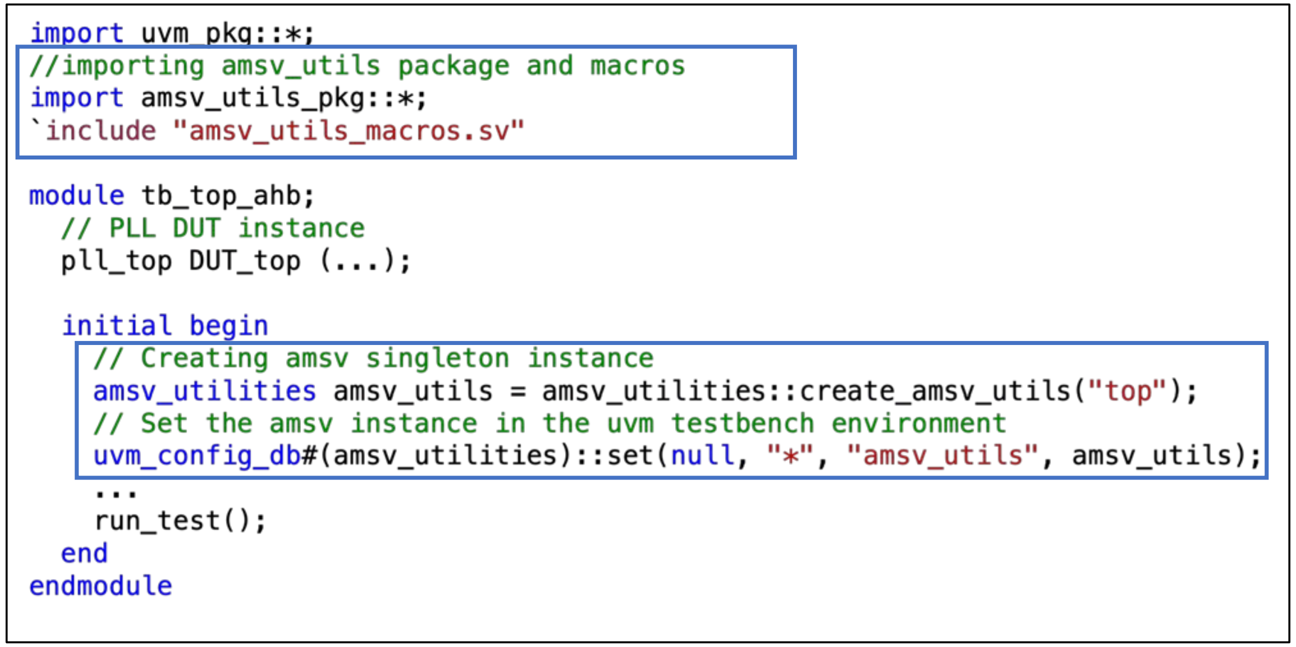
Implementing Safety-Critical Scenarios

- Testbench-controlled fault injection mechanism in the Thermal Shutdown (TSD) model
  - Synchronized with global *tsd_force_fail* parameter via *amsv_utils_sync_globals* macro
  - Bypasses shut down even as the temperature rises
  - Allows system behavior analysis under failure conditions

```
`timescale 1s/1ps
// Importing AMSV Utility Package and Macros
import amsv_utils_pkg::*;
`include "amsv_macros.sv"
// Module Declaration
module TSD(
    tsd,
    trim,
    enable,
    por_b,
    i_ptat,
    vdda,
    gnda
);
// Parameters
........................
// Port Types and Declaration
........................
// Registering model to AMSV Utilities
`amsv_utils_register("TSD", "PLL-TSD")

// Syncing with Global Variable tsd_force_fail
`amsv_utils_sync_globals(tsd_force_fail, "tsd_force_fail")

// Discreate Behavior
// Calculate temperature from i_ptat
........................

// Drive TSD
// If tsd_force_fail = 1 tie the output to 0
assign tsd = (tsd_force_fail==0) ? tsd_reg : 0;

// Analog Assertions
........................
endmodule
```

Integration into the UVM Testbench

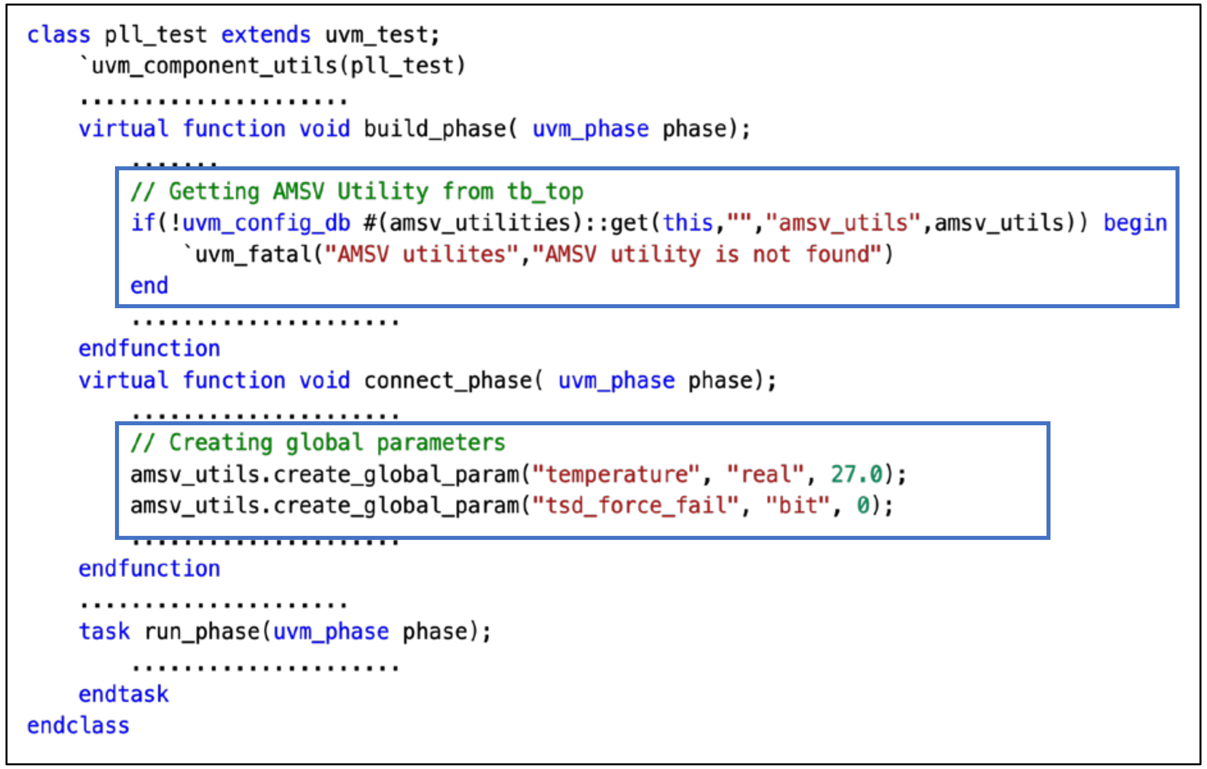
- Import *amsv_utils_pkg* at the testbench top-level
- Create a singleton instance using the *create_amsv_utils* method
- Global testbench access through UVM configuration database



```
import uvm_pkg::*;
//importing amsv_utils package and macros
import amsv_utils_pkg::*;
`include "amsv_utils_macros.sv"

module tb_top_ahb;
  // PLL DUT instance
  pll_top DUT_top (...);

  initial begin
    // Creating amsv singleton instance
    amsv_utilities amsv_utils = amsv_utilities::create_amsv_utils("top");
    // Set the amsv instance in the uvm testbench environment
    uvm_config_db#(amsv_utilities)::set(null, "*", "amsv_utils", amsv_utils);
    ...
    run_test();
  end
endmodule
```

Accessing AMSV Utilities in UVM

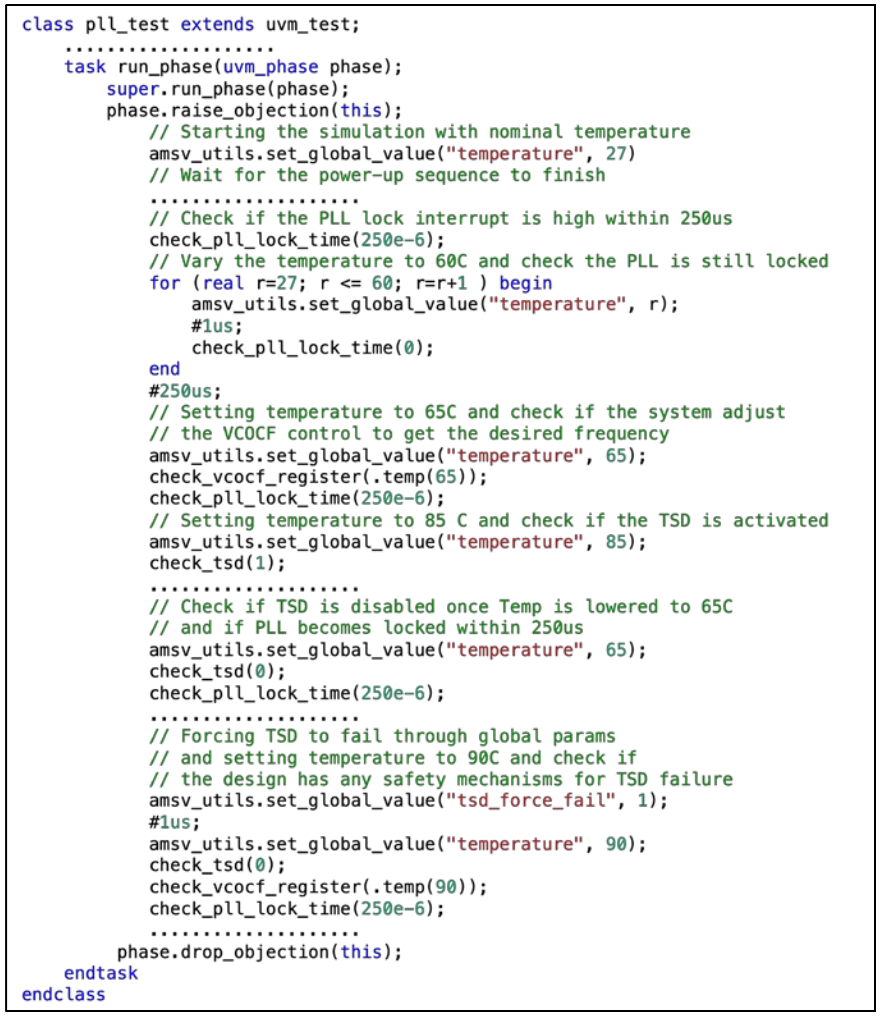
- Retrieve the AMSV instance from the UVM configuration database
- Initialize global parameters using AMSV utility methods



```
class pll_test extends uvm_test;
    `uvm_component_utils(pll_test)
    .....................
    virtual function void build_phase( uvm_phase phase);
        .......
        // Getting AMSV Utility from tb_top
        if(!uvm_config_db #(amsv_utilities)::get(this,"","amsv_utils",amsv_utils)) begin
            `uvm_fatal("AMSV utilites","AMSV utility is not found")
        end
        .....................
    endfunction
    virtual function void connect_phase( uvm_phase phase);
        .....................
        // Creating global parameters
        amsv_utils.create_global_param("temperature", "real", 27.0);
        amsv_utils.create_global_param("tsd_force_fail", "bit", 0);
        .....................
    endfunction
    .....................
    task run_phase(uvm_phase phase);
        .....................
    endtask
endclass
```

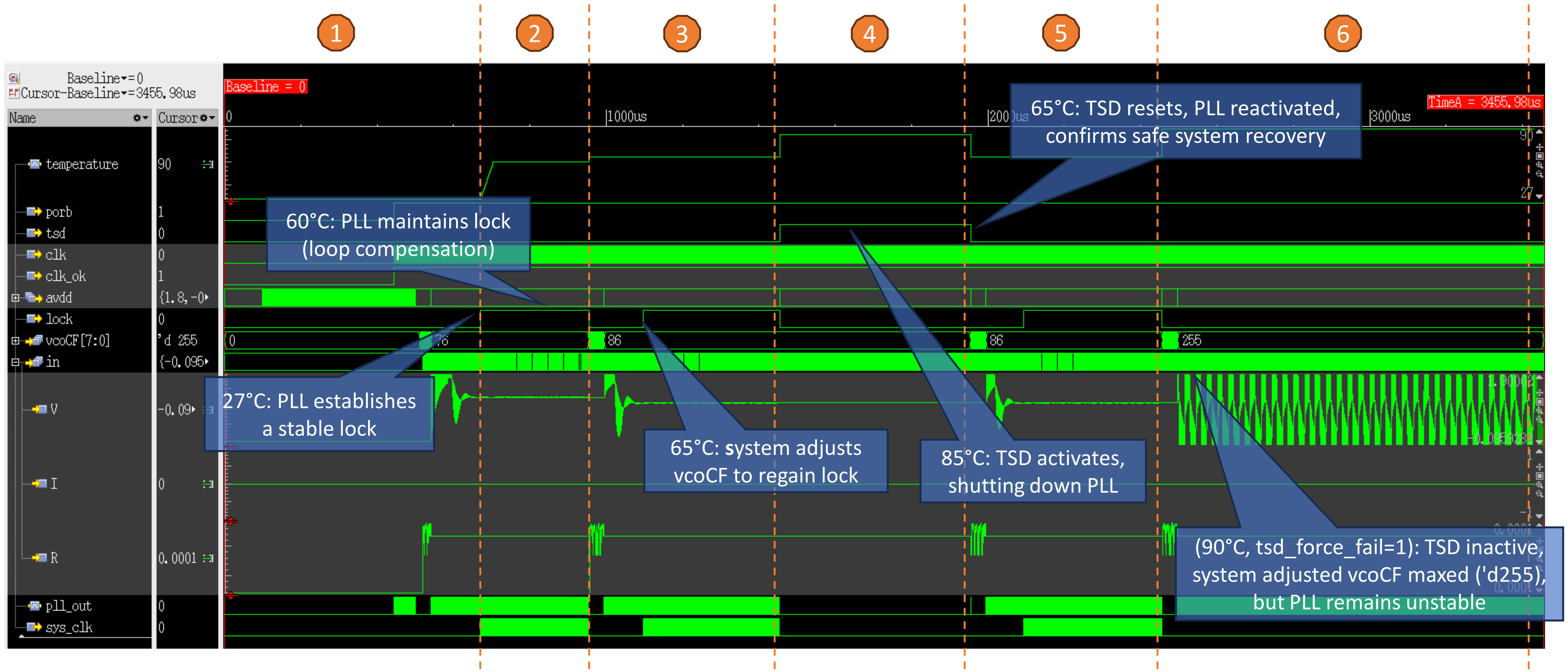
Test Cases

- Objective: AMSV utility global parameter variation and system behavior verification
- Test Scenarios & Expected Outcomes
    1. Nominal condition (27°C): PLL achieves stable lock
    2. Gradual increase (60°C): PLL maintains lock as the feedback loop compensates
    3. Rapid increase (65°C): PLL controller adjusts VCO frequency; lock achieved within 250µs
    4. Exceeding operational range (85°C): TSD block triggers shutdown, verifying protection
    5. Recovery check (65°C): PLL resumes operation, confirming proper TSD recovery
    6. TSD failure simulation (90°C, tsd_force_fail = 1): Verifies system response without TSD protection



```
class pll_test extends uvm_test;
    ....................
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
            // Starting the simulation with nominal temperature
            amsv_utils.set_global_value("temperature", 27)
            // Wait for the power-up sequence to finish
            ....................
            // Check if the PLL lock interrupt is high within 250us
            check_pll_lock_time(250e-6);
            // Vary the temperature to 60C and check the PLL is still locked
            for (real r=27; r <= 60; r=r+1 ) begin
                amsv_utils.set_global_value("temperature", r);
                #1us;
                check_pll_lock_time(0);
            end
            #250us;
            // Setting temperature to 65C and check if the system adjust
            // the VCOCF control to get the desired frequency
            amsv_utils.set_global_value("temperature", 65);
            check_vcocf_register(.temp(65));
            check_pll_lock_time(250e-6);
            // Setting temperature to 85 C and check if the TSD is activated
            amsv_utils.set_global_value("temperature", 85);
            check_tsd(1);
            ....................
            // Check if TSD is disabled once Temp is lowered to 65C
            // and if PLL becomes locked within 250us
            amsv_utils.set_global_value("temperature", 65);
            check_tsd(0);
            check_pll_lock_time(250e-6);
            ....................
            // Forcing TSD to fail through global params
            // and setting temperature to 90C and check if
            // the design has any safety mechanisms for TSD failure
            amsv_utils.set_global_value("tsd_force_fail", 1);
            #1us;
            amsv_utils.set_global_value("temperature", 90);
            check_tsd(0);
            check_vcocf_register(.temp(90));
            check_pll_lock_time(250e-6);
            ....................
         phase.drop_objection(this);
    endtask
endclass
```

Simulation Results



27°C: PLL establishes a stable lock

60°C: PLL maintains lock (loop compensation)

65°C: system adjusts vcoCF to regain lock

85°C: TSD activates, shutting down PLL

65°C: TSD resets, PLL reactivated, confirms safe system recovery

(90°C, tsd_force_fail=1): TSD inactive, system adjusted vcoCF maxed ('d255), but PLL remains unstable

AMSV Framework – Key Contributions

- Introduced as an innovative approach to mixed-signal verification
- Seamlessly integrates analog models with UVM testbenches, enabling efficient system-level verification
- Simulates real-world conditions such as temperature variations, environmental changes, and fault conditions
- Overcomes traditional verification limitations by providing speed, flexibility, and real-time parameter control beyond transistor-level simulations
- Enhances connectivity verification, system-level assertions, and coverage of analog-digital interactions
- Establishes a new standard in efficiency and accuracy, ensuring robust validation of complex mixed-signal designs

Future Directions

- Expanding capabilities
  - Future updates will include expanded model libraries for broader AMS applications
  - Support for automatic mixed-signal connectivity check
- The Road Ahead
  - A transformative shift in mixed-signal verification, driving efficiency and innovation
  - Encouraging industry-wide adoption to refine and advance verification methodologies

Questions

- Any questions