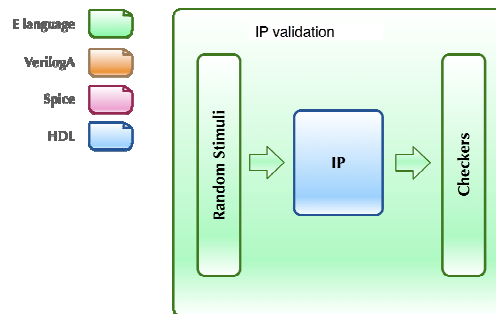


Main Functional Verification Activities Today

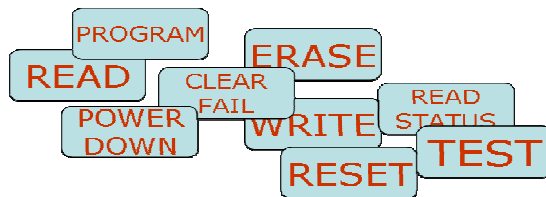


Memory Macrocells Validation

VHDL model Vs
functionality specifications Vs
RTL implementation

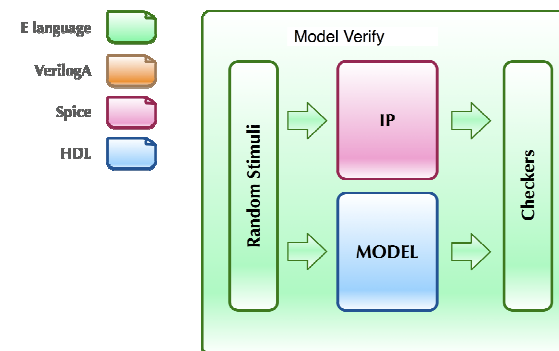


Abstraction of modes and operations



IP Model Checking

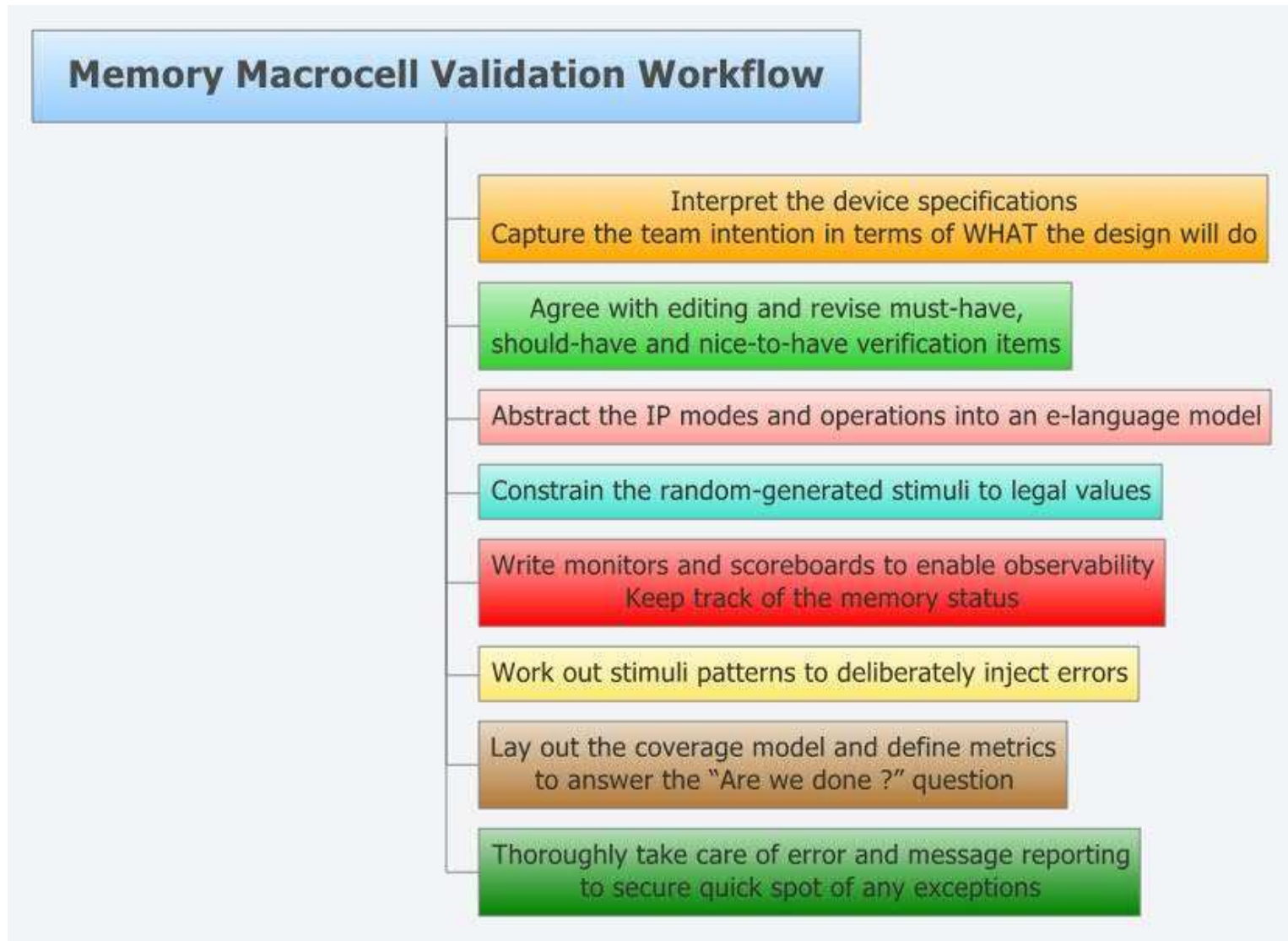
Verilog functional description
Vs SPICE netlist
Vs functionality specifications



Abstraction of electrical and timing
parameters into constraints
and coverage points

Specman-based functional validation with NCSim/HSIM/Yogitech

Memory Macrocells Validation - Today Workflow



- Automatic **random** generation of **latency** and **duration** times to test also timing violations in RTL models
 - Written systematic 'e' routines called to generate *random-constrained timed events* to shape the signal waveforms
 - *Setup/hold violations* can be issued and monitored so to test also any functional misbehaviors caused by **untimely signal assertion/de-assertion** in VHDL descriptions
- Benefiting from structuring **hierarchical tests sequences**
 - A testbench automation feature that makes available upfront **enablers** to add **flow flexibility**
 - Reduce in/out manual commenting of chunks of code
 - This **layered organization** of 'e' sequences can **promote reuse** due to its coding **modularity** and **regularity**

Memory Macrocells Validation – Testing Time Violations



Incisive Enterprise Specman Simulator

File Edit View Test Debug Tools User Help

Run some sequences with time violations generation turned on *Here's one occurring!*

```
<drive_read_parallel()> NEW TRANSACTION ITEM:
<drive_read_parallel()> [      access_kind = READ_PARALLEL].
<drive_read_parallel()> [      honesty = CORRECT].
<drive_read_parallel()> [      address (Hex)= 0x5f (Dec) 95].
<drive_read_parallel()> [enable_violation_setup_1= TRUE].
<drive_read_parallel()> [enable_violation_hold_1= TRUE].
[5167700] MACROCELL_eVC:

<drive_read_parallel> - MACROCELL_eVC MSG: STARTED READ - instr=NONE, honesty=CORREC
<read_parallel_re()> [itm.re_violation_hold= TRUE].
<read_parallel_re()> [VIOLATING hold_time= 1 (Valid >= 2)].
MACROCELL_eVC MSG, re_violation_hold_ev: 'RE' signal took in 0 [5167856] generating a H...
minimum time of T_CHRX_MIN=2ns.
[5167856] MACROCELL_eVC: MACROCELL_eVC MSG, re_violation_hold_ev: 'RE' signal took in 0 [5167856] generating a
HOLD violation against minimum time of T_CHRX_MIN=2ns.
```

Specman >

NVM_160BFSC_H

Here's how the DUT responded to the *hold* violation:

```
=====
<drive_read_parallel()> NEW TRANSACTION ITEM:
<drive_read_parallel()> [      access_kind = READ_PARALLEL].
<drive_read_parallel()> [      honesty = CORRECT].
<drive_read_parallel()> [      address (Hex)= 0x5f (Dec) 95].
<drive_read_parallel()> [enable_violation_setup_1= TRUE].
<drive_read_parallel()> [enable_violation_hold_1= TRUE].
[5167700] MACROCELL_eVC:

<drive_read_parallel> - MACROCELL_eVC MSG: STARTED READ - instr=NONE, honesty=CORRECT
<read_parallel_re()> [itm.re_violation_hold= TRUE].
<read_parallel_re()> [VIOLATING hold_time= 1 (Valid >= 2)].
MACROCELL_eVC MSG, re_violation_hold_ev: 'RE' signal took in 0 [5167856] generating a HOLD violation against minimum time of T_CHRX
[5167856] MACROCELL_eVC: MACROCELL_eVC MSG, re_violation_hold_ev: 'RE' signal took in 0 [5167856] generating a HOLD violation again
ASSERT/WARNING (time 5167856 NS) from procedure @ieee.VITAL_Timing:ReportViolation
NVM_160BFSC_HCMOS9A HOLD Low VIOLATION ON RE WITH RESPECT TO CK;
Expected := 2 NS; Observed := 1 NS; At : 5167856 NS
[5168165] MACROCELL_eVC: Read Done: READ DOUT (Hex)=0xd7; (decimal: 215) @ ADD=95 ITM DATA=215 , COMPL=40 (DIN Hex now is=0x99).
```

SimVision simulator

Memory Macrocells Validation – Hierarchical Sequences



```
extend MAIN NVM_160BFSC_HCM059A_ENV1_sequence {  
    // Boolean enabler for the chunks of tests  
    initial_reset_enabler: bool; //  
    keep soft initial_reset_enabler = TRUE; //  
    initial_powerdown_enabler: bool; //  
    keep soft initial_powerdown_enabler = TRUE; //  
    initial_memory_program_with_dir_n_compl: bool; //  
    keep soft initial_memory_program_with_dir_n_compl = TRUE; //  
    initial_memory_programming: bool; //  
    keep soft initial_memory_programming = FALSE; //  
    parallel_reading_after_the_initial_memory_program : bool; //  
    keep soft parallel_reading_after_the_initial_memory_program = TRUE; //  
    initial_block_sector_erase : bool; //  
    keep soft initial_block_sector_erase = TRUE; //  
    parallel_reading_the_erased_addresses : bool; //  
    keep soft parallel_reading_the_erased_addresses = TRUE; //  
    first_direct_programming_following_initial_erase : bool; //  
    keep soft first_direct_programming_following_initial_erase = TRUE; //  
    special_erase___involved_s03_and_s04_special_sectors : bool; //  
    keep soft special_erase___involved_s03_and_s04_special_sectors = TRUE; //  
    clear_of_fail_after_the_erase_of_the_special_sectors : bool; //  
    keep soft clear_of_fail_after_the_erase_of_the_special_sectors = TRUE; //  
    parallel_reading_after_clearing_of_fail : bool; //  
    keep soft parallel_reading_after_clearing_of_fail = TRUE; //  
    further_memory_programming : bool; //  
    keep soft further_memory_programming = TRUE; //  
    parallel_reading_after_further_memory_programming : bool; //  
    keep soft parallel_reading_after_further_memory_programming = TRUE; //  
    parallel_reading_stopping_setup_violation_on_re : bool; //  
}
```

At the top user section lie the *boolean enablers*

Written
parameterized routine
for each operational mode:

- initial_reset
- power_down
- parallel_reading
- ...

Different sequences
can call the same routine
with different parameters:

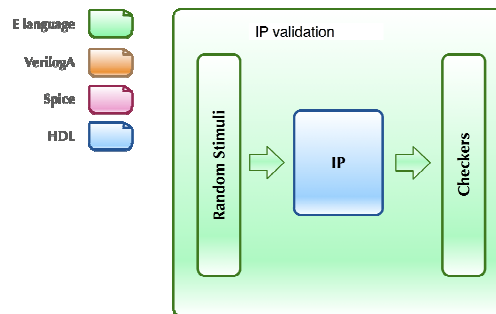
- enablers
- textual messages
- ...

Main Functional Verification Activities Today

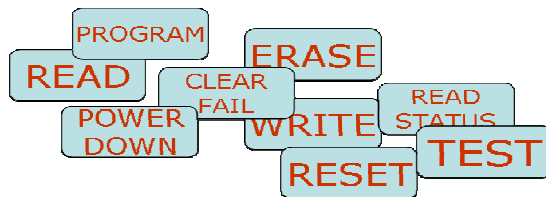


Memory Macrocells Validation

VHDL model Vs
functionality specifications Vs
RTL implementation

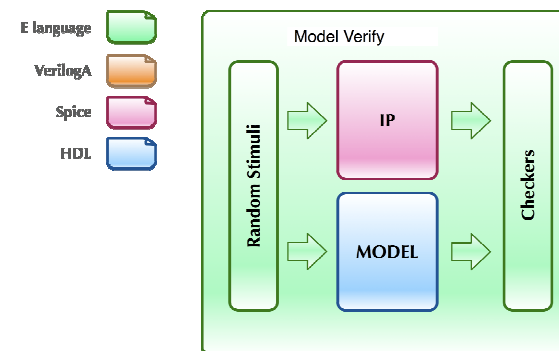


Abstraction of modes and operations



IP Model Checking

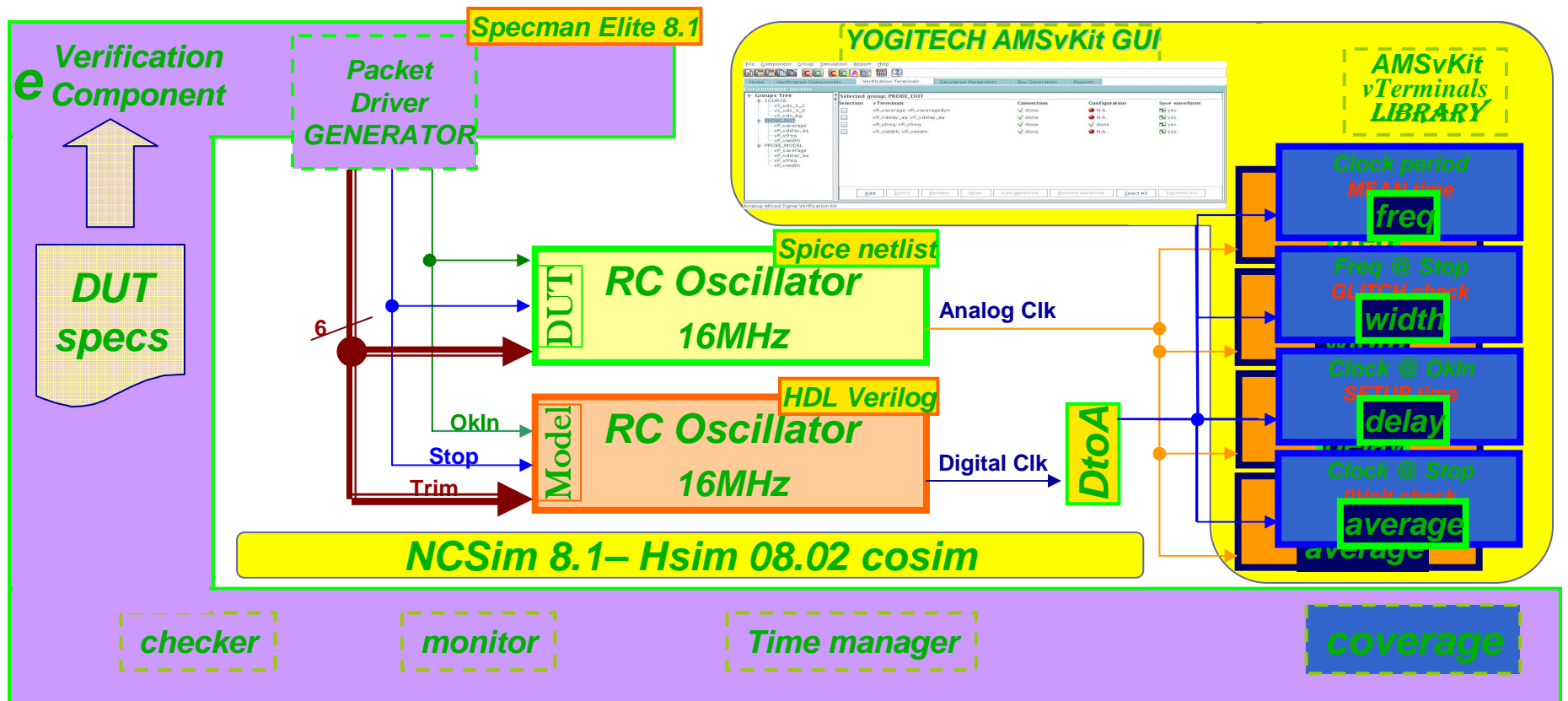
Verilog functional description
Vs SPICE netlist
Vs functionality specifications



Abstraction of electrical and timing
parameters into constraints
and coverage points

Specman-based functional validation with NCSim/HSIM/Yogitech

IP Model Checking – Today Implementation



IP Model Checking – Terminals Synchronization



Specs vs Analog IP vs Model

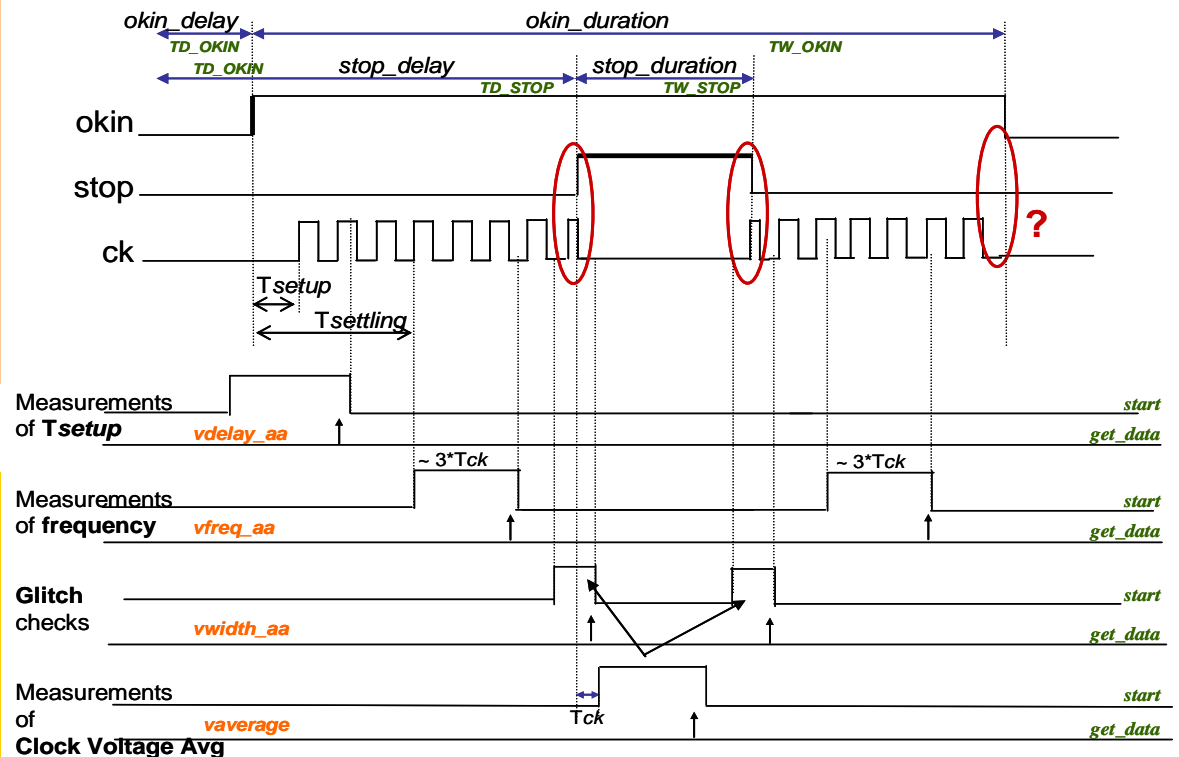
Yogitech AMS Kit and Probe Library

Oscillator features

- RC ladder technique
- RC subdue to change upon process variation
- Back-trimming compensation
- Voltage setup: Okln rises
- Stop rises: out freq = 0

Environment features

- Thresholds and PVTs as verification parameters
- Random-generated stimuli
- Constrained and direct tests



IP Model Checking - Verification Reports



Simul. time	SETUP Time	Res	FREQ BEFORE STOP	Res	FREQ AFTER STOP	Res	GLITCH Rise	GLITCH Fall	CK IDLE	Res
22100	1200	OK	13647 KHz	NONE	0 Hz	ERROR	NO	NO	18207 pV	OK
42630	1189	OK	NONE	NONE	0 Hz	ERROR	YES	YES	18207 pV	OK
57580	1189	OK	13630 KHz	NONE	0 Hz	ERROR	NO	NO	18207 pV	OK
75450	1189	OK	NONE	NONE	0 Hz	ERROR	NO	NO	18207 pV	OK
83070	1189	OK	NONE	NONE	NONE	NONE	NO	NONE	18207 pV	OK
105410	1189	OK	NONE	NONE	0 Hz	ERROR	NO	NO	18208 pV	OK
115900	1189	OK	NONE	NONE	NONE	NONE	YES	NONE	18208 pV	OK

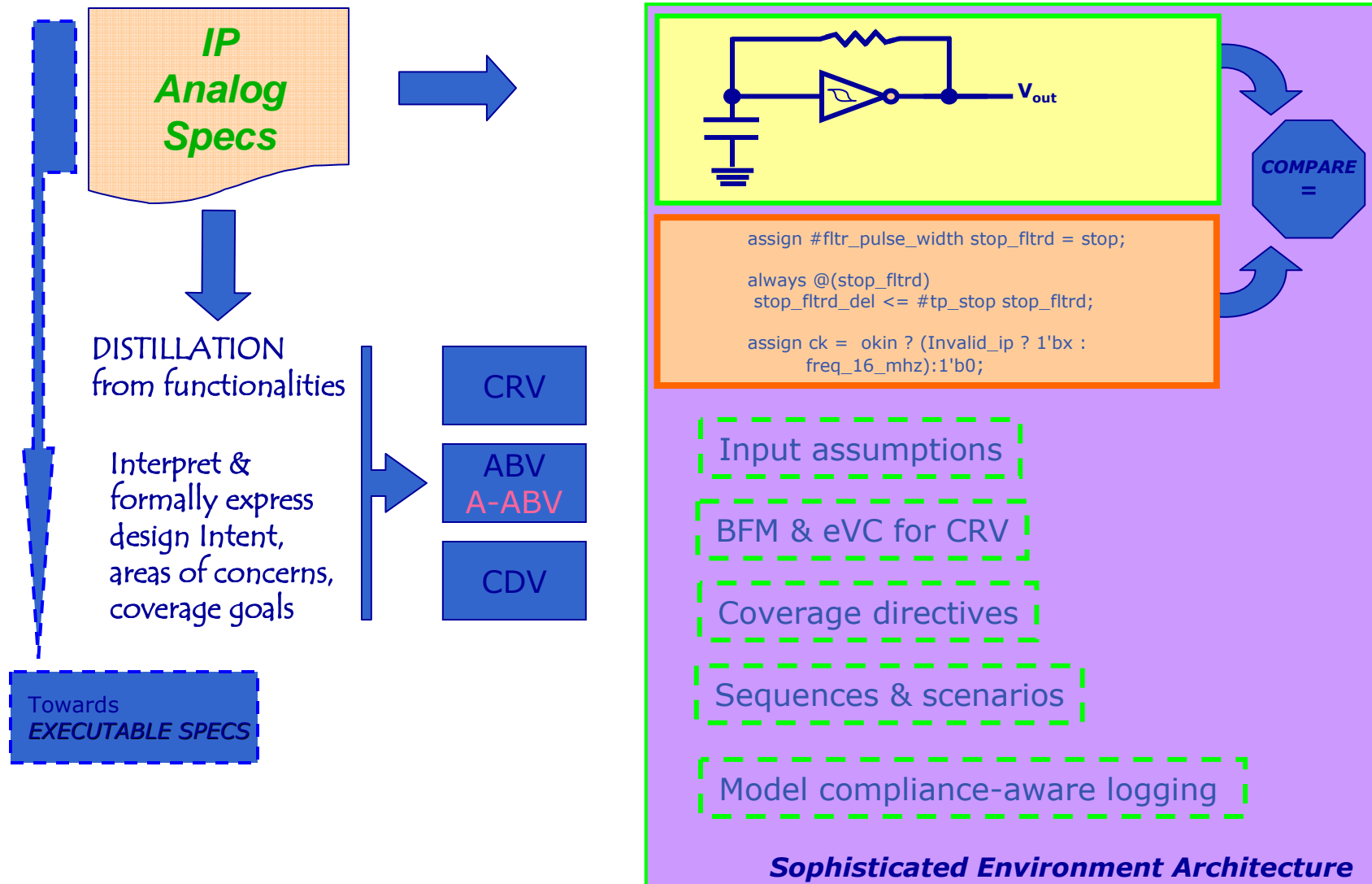
Flow benefits

- Outcome on tables speeding up final checks
- Test progress status with functional coverage stats

Flow Automation

- AMSvKit GUI assistance for:
 - Work directory setup
 - CoSim scripts generation
 - HDL wrappers generation

IP Model Checking – Possible Scenarios



- Support full co-simulation engine for AMS designs
- Broad stimuli generation strategies for coverage
- Lead to a viable formal model property checking
 - ✓ From Assertion Based Verification to Analog Formal Verification
- Design property distillation techniques
 - ✓ Derive directly from specifications all the assertions in an efficient and effective way
- Advanced logging mechanisms

Looking for ***reliable solutions*** addressing:

1. AMS Model Checking

2. Digital IP Validation

3. Analog IP Validation

4. Real Mixed-Signal Applications Validation

- Quite **limited resources** are exploring the advanced verification arena
 - ✓ An activity often perceived as "*Always Costly And Never Completed*" (J.Bergeron - Synopsys)
- The Analog/Digital **mixed expertise is rare** in design engineers and the specific know-how needed for verification often clash with required analog insights
 - ✓ Aspect Oriented Programming (AOP) – Specman
 - ✓ Object Oriented Programming – System Verilog
 - ✓ Specific languages
- Adoption of **Formal** Model Checking tools still felt awkward today
- Managers and senior designers are still reluctant to invest in novel tools
 - ✓ New paradigms can call for **different mindset** to reach higher tops

Main Goals To Attain



Driving factors leading to more advanced solutions to secure design robustness and high quality results

Higher verification productivity

Vertical and horizontal reusability

Easy access to regression capabilities

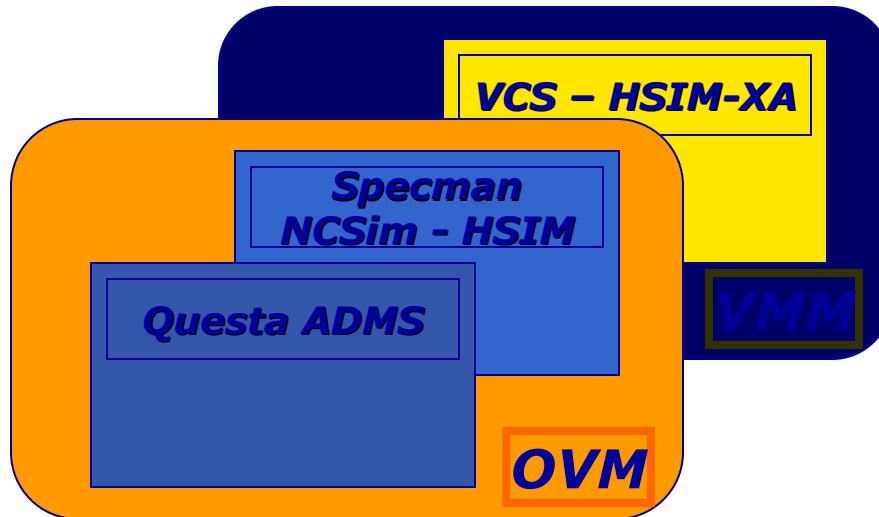
Ensured environment maintainability

Enhanced automation in sequences and scenarios generation

Flow portability over multiple platforms

Support for best practices and component libraries

AMS Model Checking – Leveraging Methodologies



*Can any available advanced methodologies be viably **extended** to take on the AMS verification challenges today ?*

Specman 'e' can still be used to:

- ✓ Abstract specs into drivers and Bus Functional Models (BFM)
- ✓ Describe **reusable** verification components
- ✓ CRV: random constrained inputs/outputs to ranges and legal values

ABV should bring in:

- ✓ Annotated assumptions on inputs in a concise form
- ✓ Express declarative descriptions as "active comments for" HDL/tests
- ✓ Capture design intents and foster early validation by spotting bugs
- ✓ Eventually leading to a rapid adoption of **formal** techniques

Topics to Address for AMS Verification



- Within the activity aimed at defining a test-bench automated flow and a verification methodology suitable to implement our AMS Model Checking tasks here are the topics to address
 - Ease of adoption and usability
 - Analog measurements enabled through digital-like mechanisms with system tasks, functions, checker libraries
 - Assertion Based Verification principles extended to detect flaws upon analog signals properties
 - The migration issue from 'e' to System Verilog
 - Parameterization and configuration of the environment
 - Example of a Frequency Check meter

- The common adoption issue has to be addressed early in order to pave the way for a wider usability
- Smart GUIs and code encapsulation could increase the degree of acceptance within the AMS designer community
- The layered organization underlying the chosen solution should enable the verification team and the designers to configure and reuse the developed environments and already constructed test-benches

- Analog measurements should be instantiated through digital-like mechanisms
- The confinement of the pure analog aspects could be achieved with callbacks asserted as properties
- An extra effort could be undertaken to port this benefits for the Real Access too
- As we envisage things to date, system tasks and functions sound to be the efficient method to implement observability of voltage/current nodes
- The availability of novel features and techniques in D-AMS are also appreciated
- Library of assertion checkers

- Assertion Based Verification principles extended to observe analog signals positively looks promising
- Design and coverage analog properties abstracted from specifications is the base technique we like to see enabled in a AMS verification ecosystem
- The directives on ***continuous signals*** should thus be monitored and checked through assertion statements in order to track down the source of the flaws as the assertion is fired upon a property or sequence that did not hold

- The migration issue is not considered a real plus since our current verification environments are coded in 'e' language
- System Verilog language should be preferred against PSL due to its standardization and wider support
- Nevertheless we are currently aware about a few assets that PSL can boast like more expressive timed expressions and cross-language usage

Parameterization and Configuration



- We are also after a full parameterization and configuration of the available analog probes. This goal could be investigated by looking at the support of the **bind** mechanism
- A bind-like construct could provide a suitable argument-passing mechanism to either specify the node pathname or pass in terminal parameters
- The externally configured probe must be configured at its invocation with value ranges, thresholds and tolerances to undercover possible out-of-range and non-compliances
- The list of analog measurements observed through monitors and checkers should at least comprise the following early detecting functions
 - Period or frequency consistency of continuous signals over a given monitored time interval
 - Possible presence of clipped signals
 - Average voltage and current
 - Peak power consumption

Example of Frequency Check Probe



```
property CHECK_FREQ_IS_IN_RANGE (trigger_sig, freq, freqMin,
    freqMax);
    @(posedge trigger_sig)
        <enable_sig> |-> ( (freq >= freqMin) && (freq <=
freqMax) );
endproperty
```

```
period_prop: assert property
    ( CHECK_FREQ_IS_IN_RANGE(<trig_sig>, period, min_period,
    max_period) )
    $display("\nFreq in range (Min Req period='%t', Max Req Period='%t').",
    min_period, max_period);
    else begin
        $error(0, "\n %m Freq out of range! Min Req period='%t', Max Req
    Period='%t'\nCaptured period='%t'\ntrigger_sig=%d; top_in1=%d, top_in2=%d.\n",
    min_period, max_period, period, xor_out, top_in1, top_in2); //
    end
```

Example of Frequency Check Probe



```
// sig_get_freq made available either as system task or function or module
function time sig_get_freq(input <HIER_ANALOG_node>, input
    real thMin, input real thMax); // Add more config arguments...
    <IMPLEMENTED BY EDA VENDOR>
endfunction
```

```
task grab_freq; // Calling the System Task/Function
    output time generatedFreq; input an_sig; input real thMin,
    thMax; // Add more config arguments...
    begin
        generatedFreq = sig_get_freq(an_sig, thMin, thMax); //
    end
endtask
```

```
always @( <trig_sig> ) // whenever the signal changes
    begin
        grab_freq(period, <hier_an_sig>, <thres_min>,
        <thres_max>); // Add more config arguments...
    end
```