Accelerated SOC Verification Using UVM Methodology for a Mixed-Signal Low Power Design

Giuseppe Scata – Texas Instruments (g-scata@ti.com)
Ashwini Padoor – Texas Instruments (ashwini.padoor@ti.com)
Vladimir Milosevic – ELSYS Eastern Europe (v-milosevic@ti.com)

Overview

- **Introduction**
  - Unified SoC flow for directed and constrained random verification approach
- **DUT Overview**
  - DUT Overview – I am Mixed Signal
- **SoC TB Architecture Overview**
  - SoC Testbench: An Overview
  - SoC Testbench: Stimuli
  - SoC Testbench: Direct Testing
  - SoC Testbench: TB/SW synchronization
  - IPDV Reuse Examples
  - uVC: Use-Model
  - Checkers and Assertions
  - Functional Coverage Tracking
  - Test End Mechanisms
- **Going AMS**
- **Results & Observations**

The Motivations

The microcontroller **SoC verification strategy refinement** to the next level:

- **Unified flow** from RTL to AMS
- Reduction of development cycles via **reuse of IP-DV components**
- Uncover each possible design surprise by using **always on checkers / scoreboards**
- Adherence to Industry standard methodology: **UVM**
- Support for **direct** and **random** verification approaches

© Accellera Systems Initiative

DUT – I am Mixed Signal

- Design is not just digital: Analog On Top (**AOT**) **Low Power Mixed Signal** Design;
- The toplevel is an **analog netlist** instantiating Real Number Models and the digital top.
- Netlist is **automatically generated** by analog tool netlister – The tool extracts the design hierarchy details until it finds a Verilog view of a cell.

© Accellera Systems Initiative

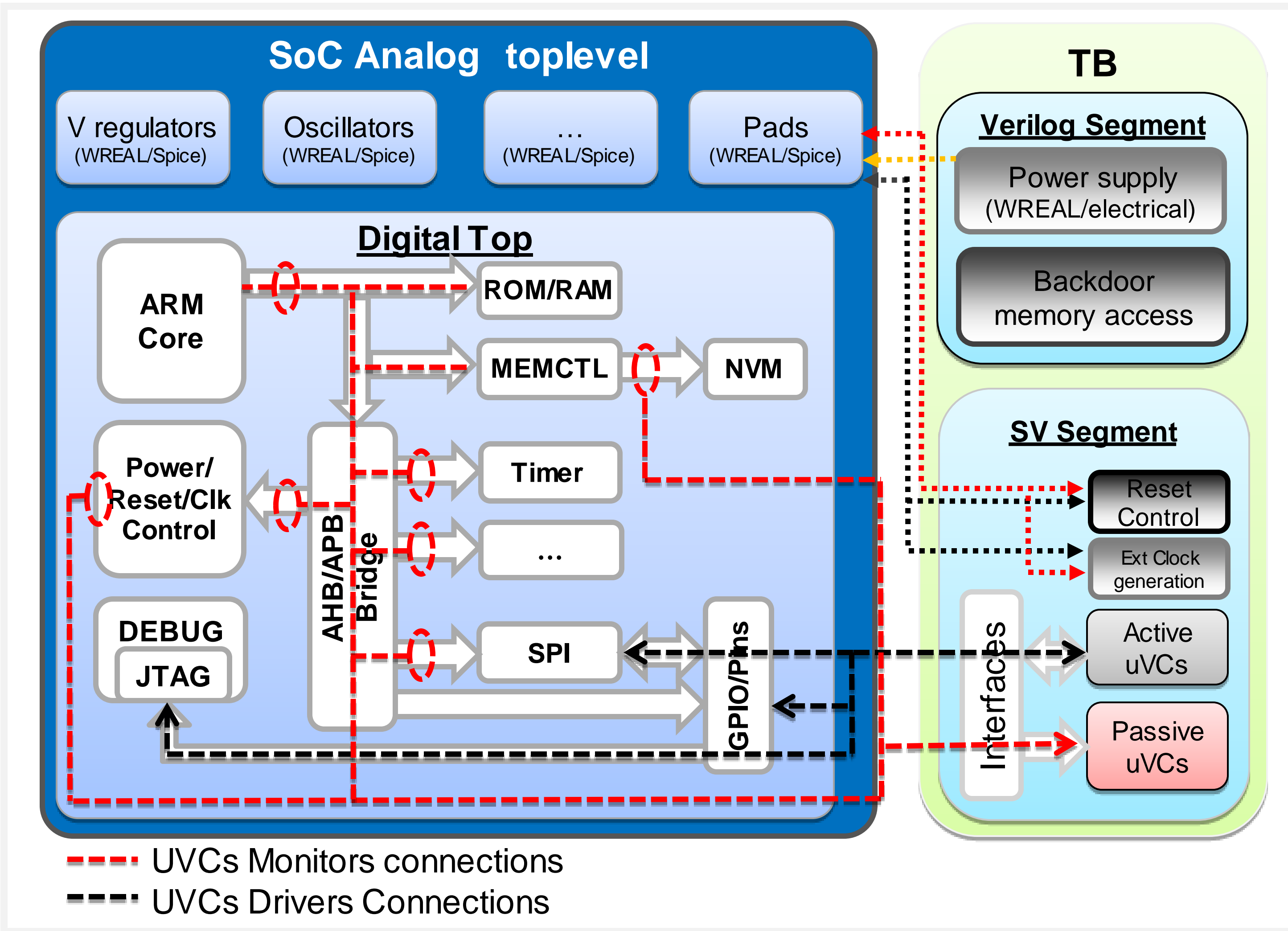
SoC Testbench: An Overview

The testbench framework contains the following components:

- **Verilog segment**: Plain Verilog part;
- **System Verilog** segment: UVM compliant components;
- **Hardware/Software** synchronization logic;

accellera SYSTEMS INITIATIVE
2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

SoC Testbench: An Overview



SOC Testbench: Stimuli

The testcase definition constitutes:

- A **System Verilog file** that defines a SV Sequence;
- A **Software** file (C/ASM/memory image) which defines the data which needs to be loaded into the SoC memory;
- Optional **support files** (linker command files, configuration files, …);

SOC Testbench: Stimuli

Universal Verification Component (uVC) deployment for generating the traffic :

- SOC external Stimuli: The stimulus generation on external I/O **via the uVCs;**
- SOC software Stimuli: **uVC based constrained random software data** handling using mailbox mechanisms;
- Handling Stimulus on multiple interfaces in a controlled way: multiple uVC integration using **Virtual sequencer;**

© Accellera Systems Initiative

SoC Testbench: Direct Testing

Allow "*legacy*" direct verification methodology in Verilog style via instantiating a dummy sequence that just waits for the "*test end*" event :

```
initial begin
// Verilog direct test
// paired with a C file
…
// testend event via SW
// or via task call
end
```

```
class `tc_name extends uvm_test;
 soc_tb ve;
 `uvm_component_utils(`tc_name)
 function new(…);
  super.new(name,parent);
 endfunction
…
virtual function void build_phase(uvm_phase phase);
 super.build_phase(phase);
 uvm_config_db#(uvm_object_wrapper)::set(this,
   "ve.virtual_sequencer.run_phase",
   "default_sequence", sw_seq::type_id::get());
endclass : `tc_name
```

SoC Testbench: TB/SW Synchronization

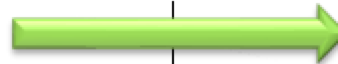
```
int main ()
{
 while (1) {

  // synchronize with TB
  TB_synch = 1;

  // Use random data got from TB
  switch (DATA_MAILBOX1) {
   case 1: fct1 (DATA_MAILBOX2); break;
   case 2: fct2 ();              break;
   case 3: fct3 (DATA_MAILBOX3); break;
   case 4: test_end();           break;
   default: ERROR++;
 }
}

void fct1 (int);
void fct2 (void);
void fct3 (uint32_t);
```

C-test issues a write to dedicated memory location with a predefined key



Testbench

Testbench detects a write to TB_synch's address and generates an event according to its value;

uVC sequence

uVC reacts to the testbench's event by storing random constrained data into the device's memory (DATA_MAILBOX*) thus achieving randomization of the C-code execution.

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

IP-DV uVC Reuse

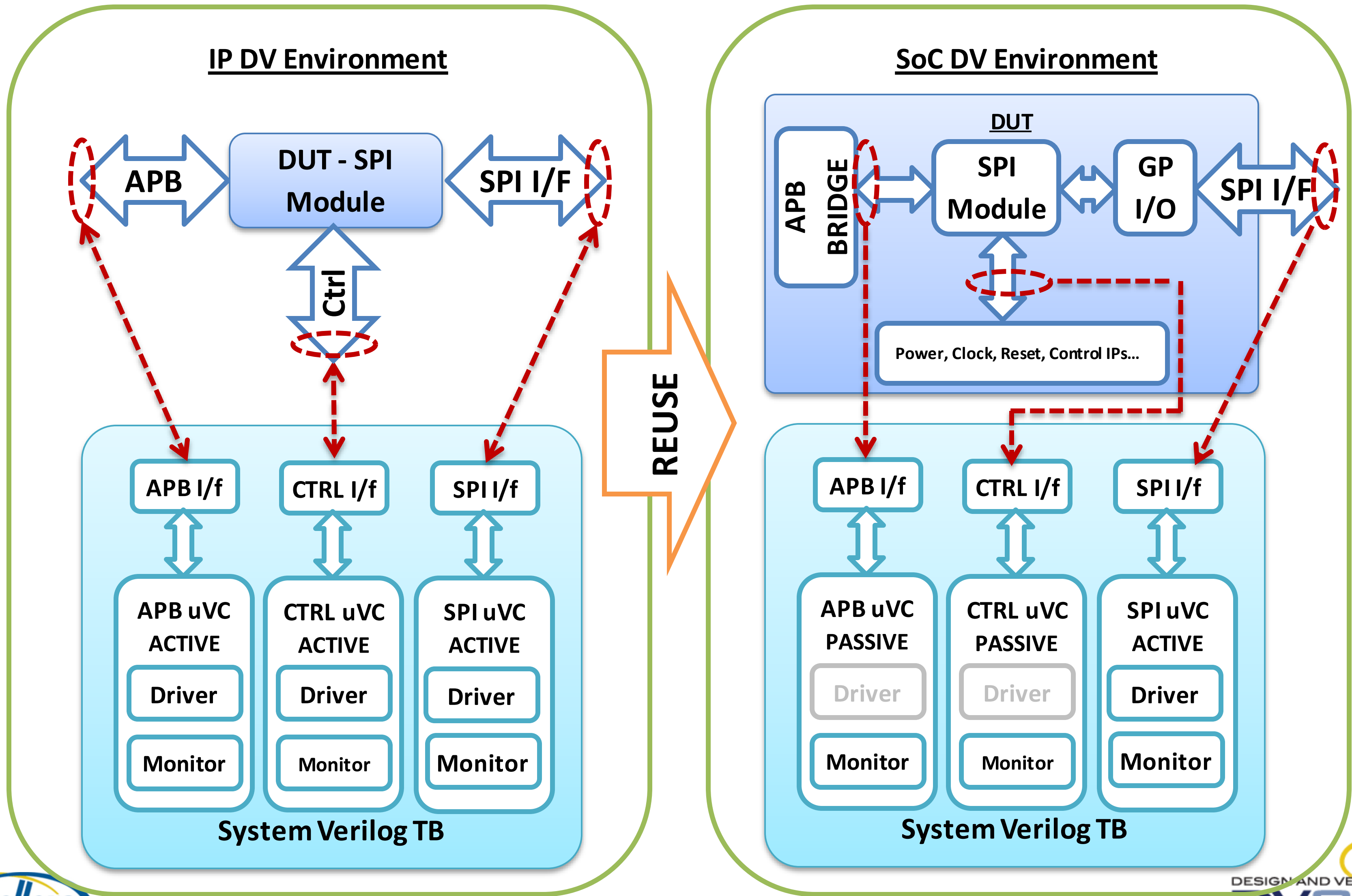
Reused components:

- Any **passive** component from IP DV – monitors:
  - Functional coverage definitions;
  - Protocol checkers;
- Any **active** component - driver that drives **external I/Os to the DUT**:
  - In SoC environment drivers are connected to the DUT boundary (Example:Serial interfaces uVC);

Redundant components:

- Any **active** component that drives signals **internal to the SOC:**
  - Interconnect signals (APB, AHB) – **use C code to generate stimuli;**
  - Signals connected to other IPs (Clock, Power, Reset, etc.) – **use specific scenarios** to trigger these signals by various SOC internal blocks;

IP-DV uVC Reuse



uVC: Use-Model

- Scoreboards are **automatically configured** according to the transactions based on IP configuration through the register interfaces;

- Availability to check design behavior automatically **reduces** testcase **development effort**;

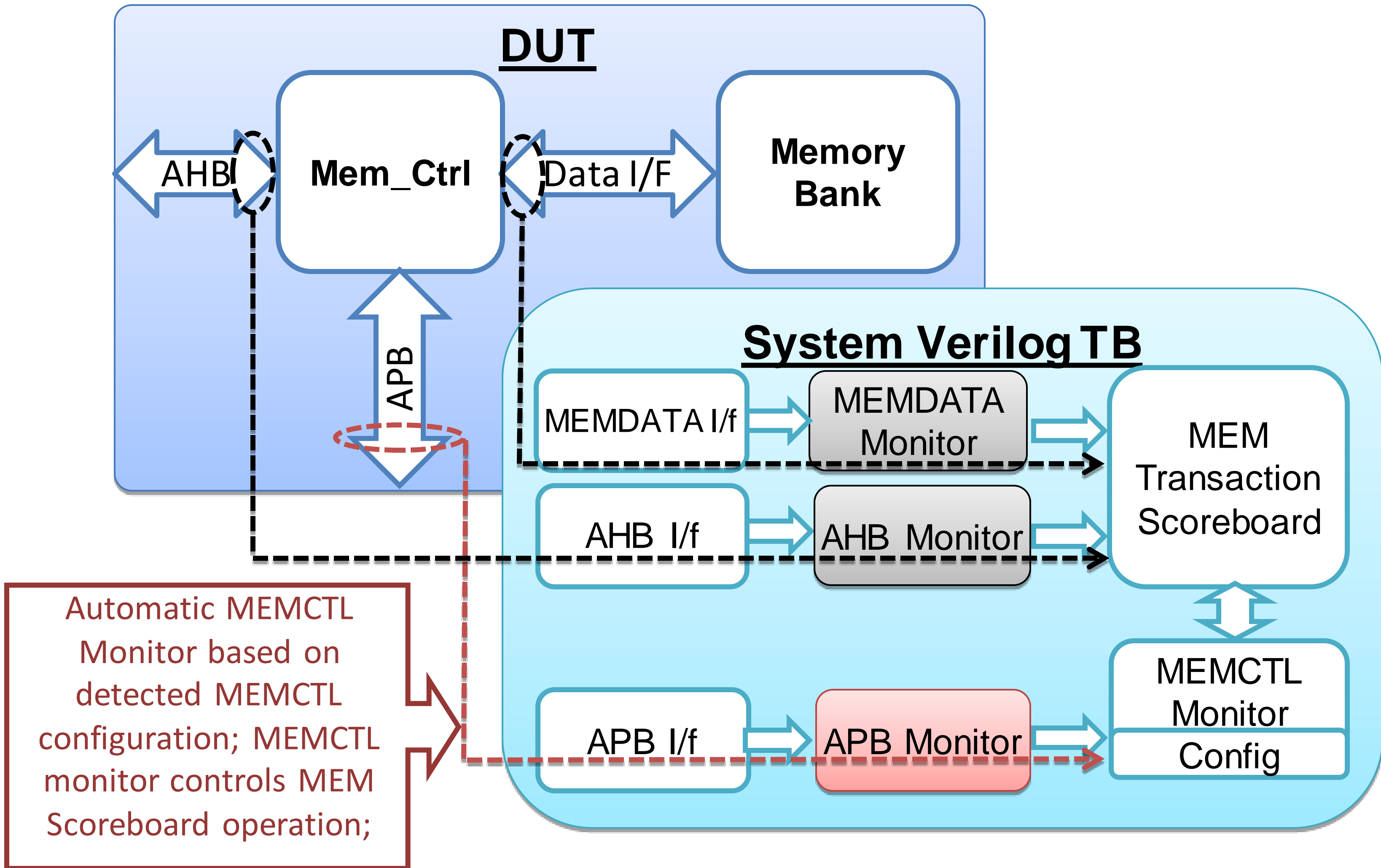
- Checkers always are enabled for any testcase, and **even if application software** is run;

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

uVC: Use-Model



© Accellera Systems Initiative

Checkers and Assertions

Checkers are implemented at various design abstraction levels to ensure design correctness

Data Checkers

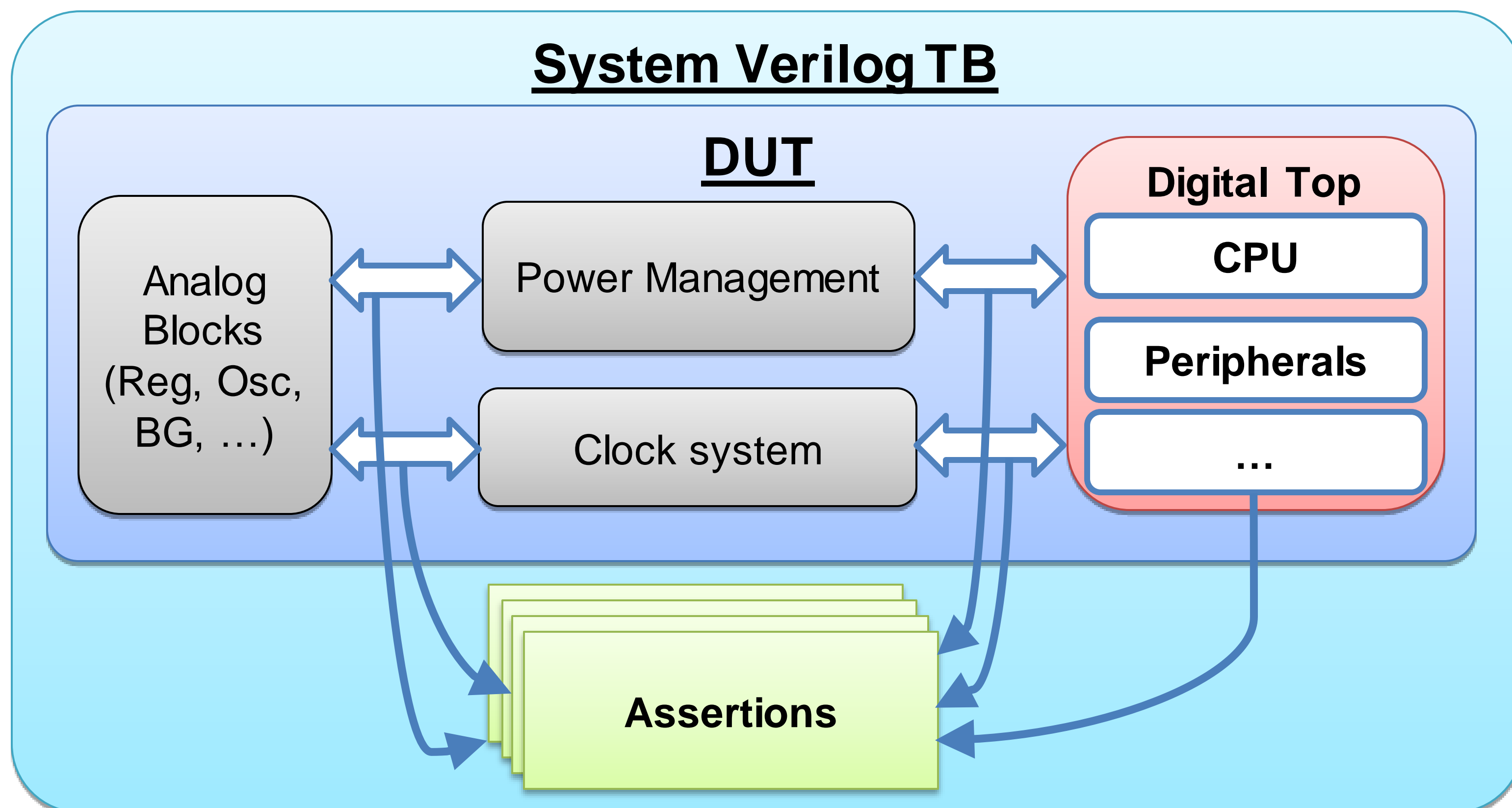
Protocol Checkers

Inbuilt Self Checking Mechanisms

© Accellera Systems Initiative

Checkers and Assertions

Always running assertions based checkers are implemented to ensure the clock and power management block functionality correctness at system level.



Checkers and Assertions

```
task ams_regulator_chk();
 forever begin
 @(posedge vif.tbclk iff (vif.resetall === 0 &&
   cfg.checkers_en == 1 && (vif.vdd_lvl < `VDD_LVL_MIN)))
    nRST_AMS_CHK: assert(vif.pmucore_poresetn_i == 1'b0)
   else begin
       `uvm_error(get_type_name(),
       $psprintf("CORE_RESET: %1b,not expected (low Reg
       Voltage LvL) @ %0t!\n", vif.pmucore_porstn, $time))
   end
```

```
// If RESET_PIN is risen, in next cycle RESET_SD and RESET_SYS has to be asserted
property reset_pin_asserted;
 disable iff (checkers_en == 0 || sig_reset == 0)
 $rose(sig_reset_pin) |-> ##1 (~sig_reset_sdn & ~sig_rst_sysn);
endproperty

assert property (reset_pin_asserted)
 else begin
 `uvm_error("RSTCTRL_IF", $sformatf("\n Reset pin should
  assert RESET_SD and RESET_SYS in next cycle @ %0t!\n", $time))
end
```

Checkers and Assertions

- Standard **UVM Checkers** for protocol checking
- **UVM Scoreboard implementation** for data paths
- **Software checkers** for register / some direct tests

Functional Coverage Tracking

- **Coverage** groups in uVC Monitors – standard uVM method;
- **VITAGs** – Verification Item Tags:
  - Implemented in TB as **SV Assertions;**
  - **Manually triggered** to indicate PASS/FAIL status:
    - C code - library function call;
    - Verilog testcase - dedicated task call;
  - Used where automatic checkers cannot be used
  - Track a **directed scenario** or any **part of a directed scenario;**

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Test End Mechanisms

The testcase end declaration:

- uVC sequence end;
- Software: calling a “test_end()” function;
- Verilog (direct testing): generating a “test end” event;

Pass / Fail criteria:

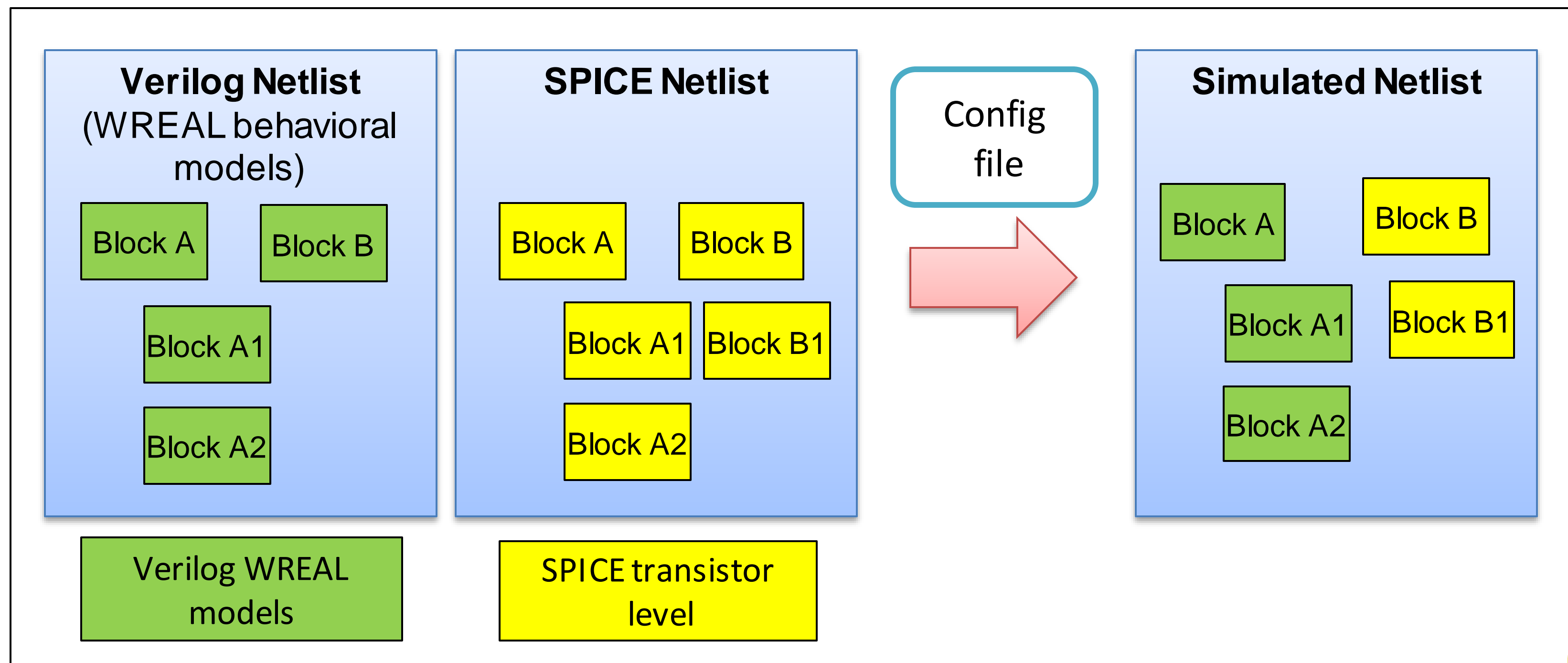
- No uVCs errors;
- No assertions errors;
- No software errors (error counter in memory);
- At least one success event;

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Going AMS

- Allows to use of the goodness of the digital verification flow;
- Possibility to mix abstraction level of analog blocks (models/transistor level) via configuration file;



Results and Observations

The UVM based verification infrastructure helped us:

- To have a single verification environment for digital, AMS and software validation;
- Helped us to overcome IP verification gaps;
- To maximize verification quality in a very minimal schedule;

The silicon validation is showing positive results

© Accellera Systems Initiative

Questions ?

Thank you !