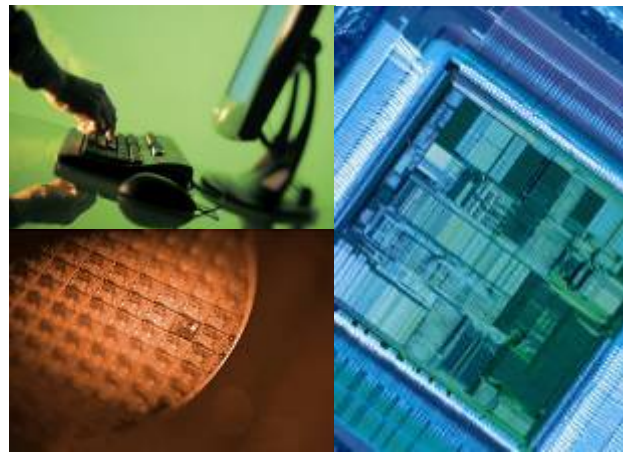


# VMM: Proven Approach for Verification with SystemVerilog

Interoperability  
Developers' Forum  
October 2007



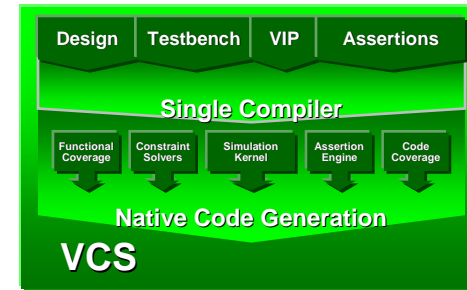
# Agenda

- Functional Verification Focus from Synopsys
- VMM overview: Using SystemVerilog constructs
- Why VMM Methodology
- Examining VMM base classes
  - vmm\_env base class
- VMM usage in design-verification environments
- Summary

# Functional Verification Focus

## VCS® and Magellan™

Performance & technology leadership



## SystemVerilog & VMM Methodology

300+ SystemVerilog customers  
Premier & proven methodology  
Verification Predictability

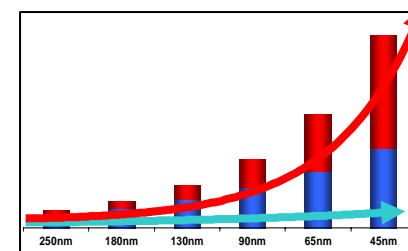


Plan Capture & View	Register Abstraction	VMM Checker
Plan Report	Hardware Abstraction	VMM - SystemC TLU
Plan Annotation	Memory Allocation	Test Composer
Spreadsheet Interface	Data Stream Scoreboard	VMM - Aware Debug
Planning Methodology	Reusable Environment	Environment Composer

VCS Functional Verification Solution  
SystemVerilog • SystemC • VHDL • Questa • Native Testbench •  
Constraint Solvers • Assertion Analysis • Unified Coverage • Unified Debug

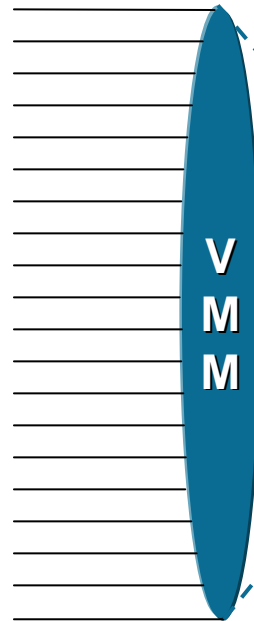
## Power-Aware Verification

End-to-end low-power solution



# VMM Harnesses The Power Of SystemVerilog

Coverage Groups    Classes  
Events    Virtual Methods  
Assertions    Coverage-Driven  
Inheritance Abstraction  
Random Generation    Interfaces  
                          Properties    Clocking Blocks  
Solver    Constraints  
Coverage Points    Messages  
Modports    Self-Checking  
                          Program Blocks



*State-of-the Art Verification*

- **Premier, Proven Methodology**
- **Used by 100s of teams, globally**
- **Growing ecosystem of books, training, services, tools, IP**
- **VMM Standard Library provided with VCS**
  - Source code at no extra charge
  - IEEE 1800 compliant



# What VMM Delivers

## Clear Guidelines

- ✓ 463 Rules
- ✓ 510 Recommendations
- ✓ 383 Suggestions

**Compliance**

## Base Classes

- ✓ Data models
- ✓ Transactors
- ✓ Verification Environments

**Consistency**

## Utilities

- ✓ Message Service
- ✓ Notification Service
- ✓ Transaction-Level Interface

**Scalability**

## Pre-Defined Functions

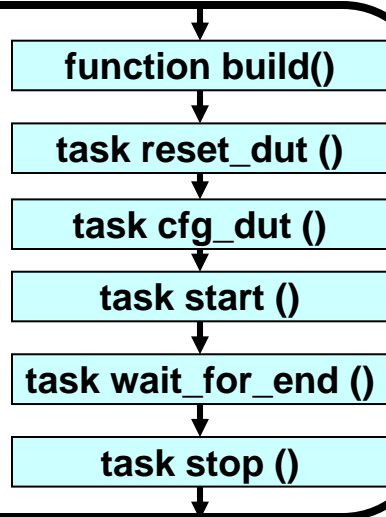
- ✓ Assertions
- ✓ Random Generators
- ✓ Coding Templates

**Productivity**

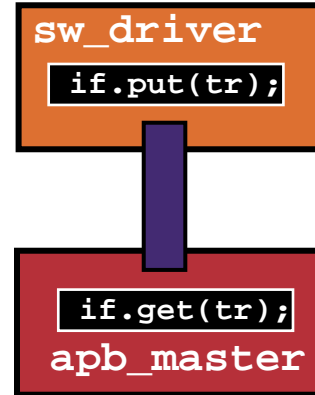
# VMM base Class Libraries

## Enhance Productivity, Enable Interoperability

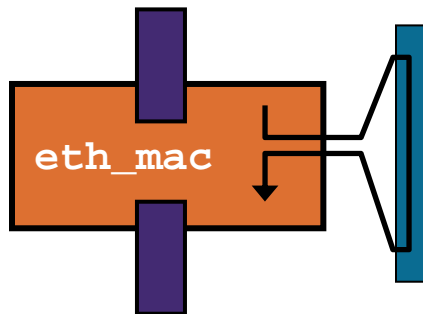
Flow Management



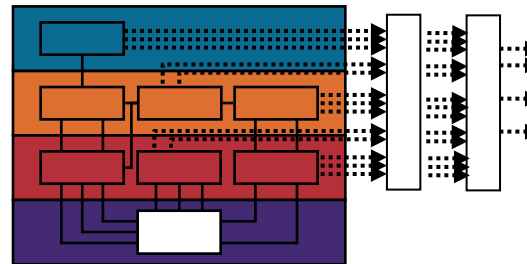
Transaction-Level Interface



Reusable Transactors

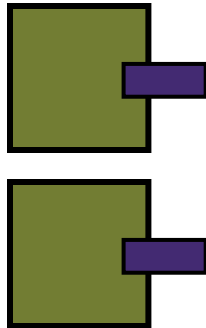


Messaging Service

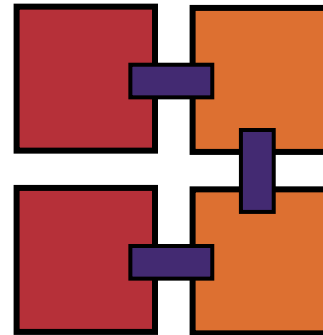


# Additional automation via VMM

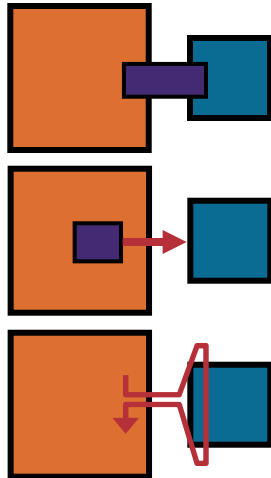
Random  
Generators



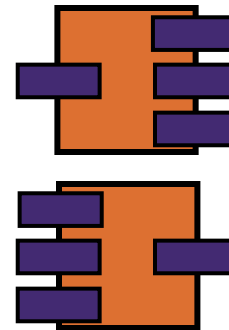
Transaction-Level  
Modeling



Functional  
Coverage



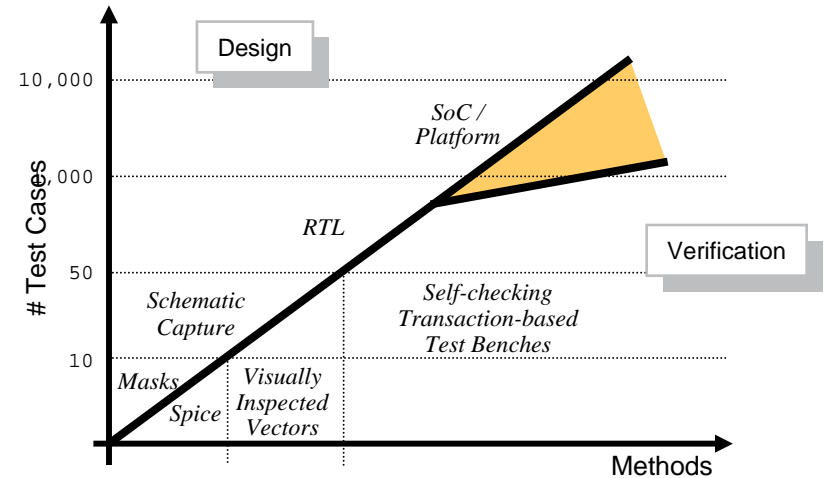
Broadcast &  
Scheduler



# Why VMM testbench methodology?

## Addressing verification complexity

- Applicable to all design styles
- Proven in many large projects and customers
- VMM helps design verification teams with:
  - Ramping-up on SystemVerilog.
  - Adopt the constrained-random verification
  - Increase design coverage and quality
- Quickly start the first tests running
- Quickly find bugs in the design.
- Promote creation of reusable
  - Data models
  - Transactors
  - Generators
- Use testbenches across levels, module to full chip



# VMM and VMM based verification: Field-proven

**“... used the VMM to help us create a well-structured, scalable testbench environment..”  
Simon Lacroix, Ross Video Limited**

**“VMM standard library provides a ready-to-run scenario generator that can be customized to provide structured streams of randomized stimulus for a verification environment.”  
Jonathan Bromley, Doulos Ltd.**

**“The VMM methodology was of immeasurable help in getting us started with SystemVerilog and has enabled our team to build a complete, robust and scalable verification environment in just a matter of months.”  
Scott Scheeler, Enterasys**

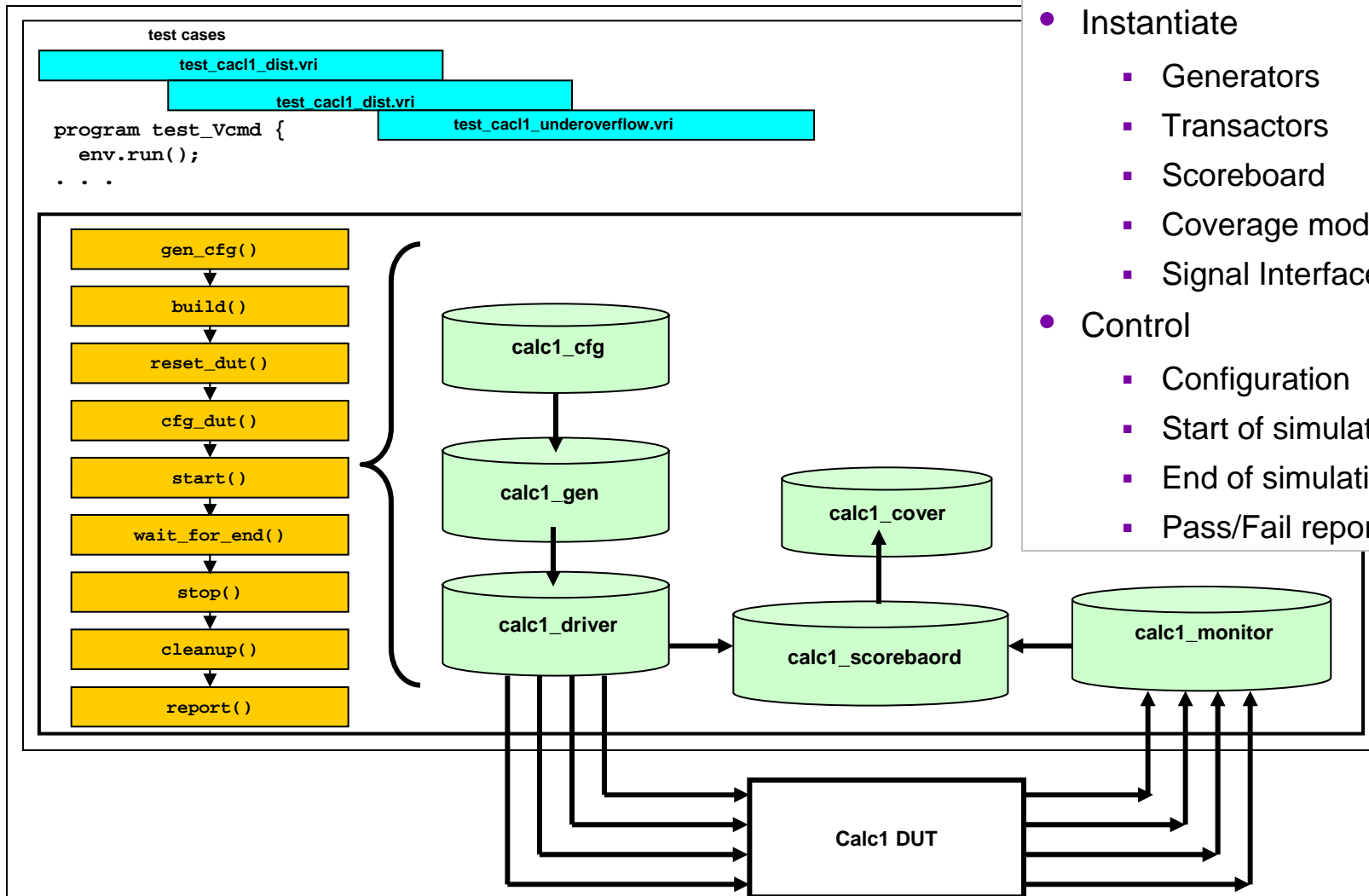
**“Integration of the VMM VIP with our design proved to be straight forward and within a few days were able to shake the DUT off of its bugs. Another week was spent in fine tuning the custom transactor with its self checking structure and functional coverage model”  
Junfeng Wang, Broadcom**

**“VMM provides an excellent starting point to create robust, efficient and re-usable verification environment”  
Hans van der Schoot, Xtreme EDA**

# Verification Environment (1)

## vmm\_env class, top level testbench

- Encapsulate verification environment
  - Reused by all tests
- Instantiate
  - Generators
  - Transactors
  - Scoreboard
  - Coverage model
  - Signal Interfaces
- Control
  - Configuration
  - Start of simulation
  - End of simulation
  - Pass/Fail report



## ***Verification Environment Class: Key Benefits (2)***

### ***vmm\_env***

- Environment Creation takes less time
  - Majority of environment code is included in VMM
  - Best practices are included in classes and examples
- Testbench Integration is simplified
  - All IP blocks have the same execution flow
  - All transactors start and stop in the same way
- Enables IP reuse
  - Environment, Tests and IP are clearly separated
  - Code (VIP) is created using reusable classes
- Legal Device Configurations are tested
  - Regression can select different DUT configurations
  - Configuration object is randomized and constrained

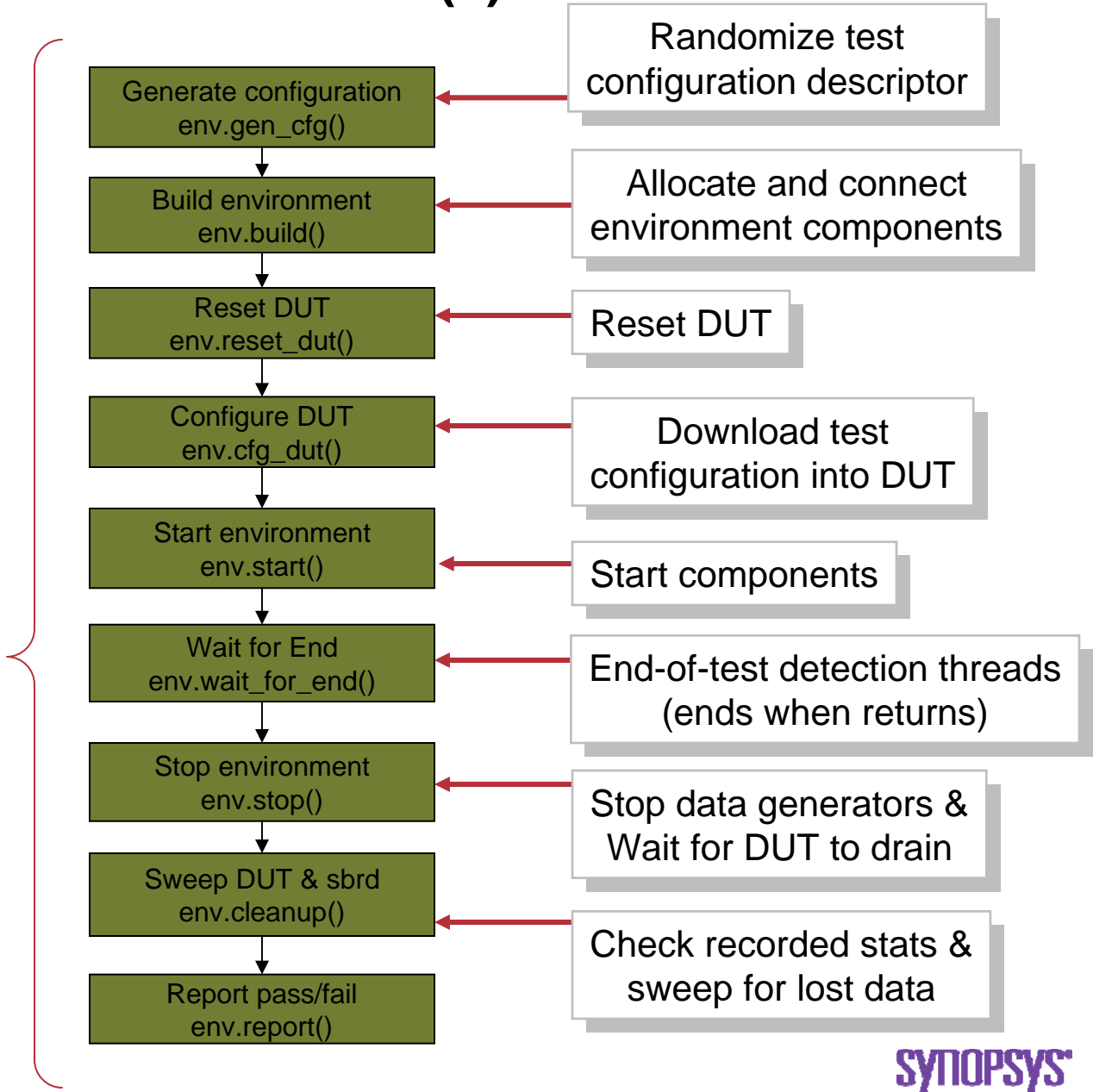
# Verification Environment (3)

## Execution Flow

- Sequence of virtual methods
- Specialized for the DUT

env.run()

Extend to implement DUT-specific verification environment



# Verification Environment (4)

## Execution Flow Test extensions

- Simplest test

```
program test;  
  initial begin  
    verific_env env = new(...);  
    env.run();  
  end  
endprogram
```

- Initial debug, trivial test

```
program test;  
  verific_env env = new(...);  
  
  initial begin  
    env.gen_cfg();  
    env.rand_cfg.frames_nb = 1;  
    env.run();  
  end  
endprogram
```

- Corner-case test

```
class my_eth_fr extends eth_frame;  
  rand bit [47:0] mac_address;  
  constraint one_port_only {  
    da == mac_address;  
  }  
endclass  
  
program test;  
  verific_env env = new();  
  initial begin  
    env.build();  
    begin  
      my_eth_fr my_fr = new;  
      env.src[0].rand_fr = my_fr;  
    end  
    env.run();  
  end  
endprogram
```

# VMM usage in industry

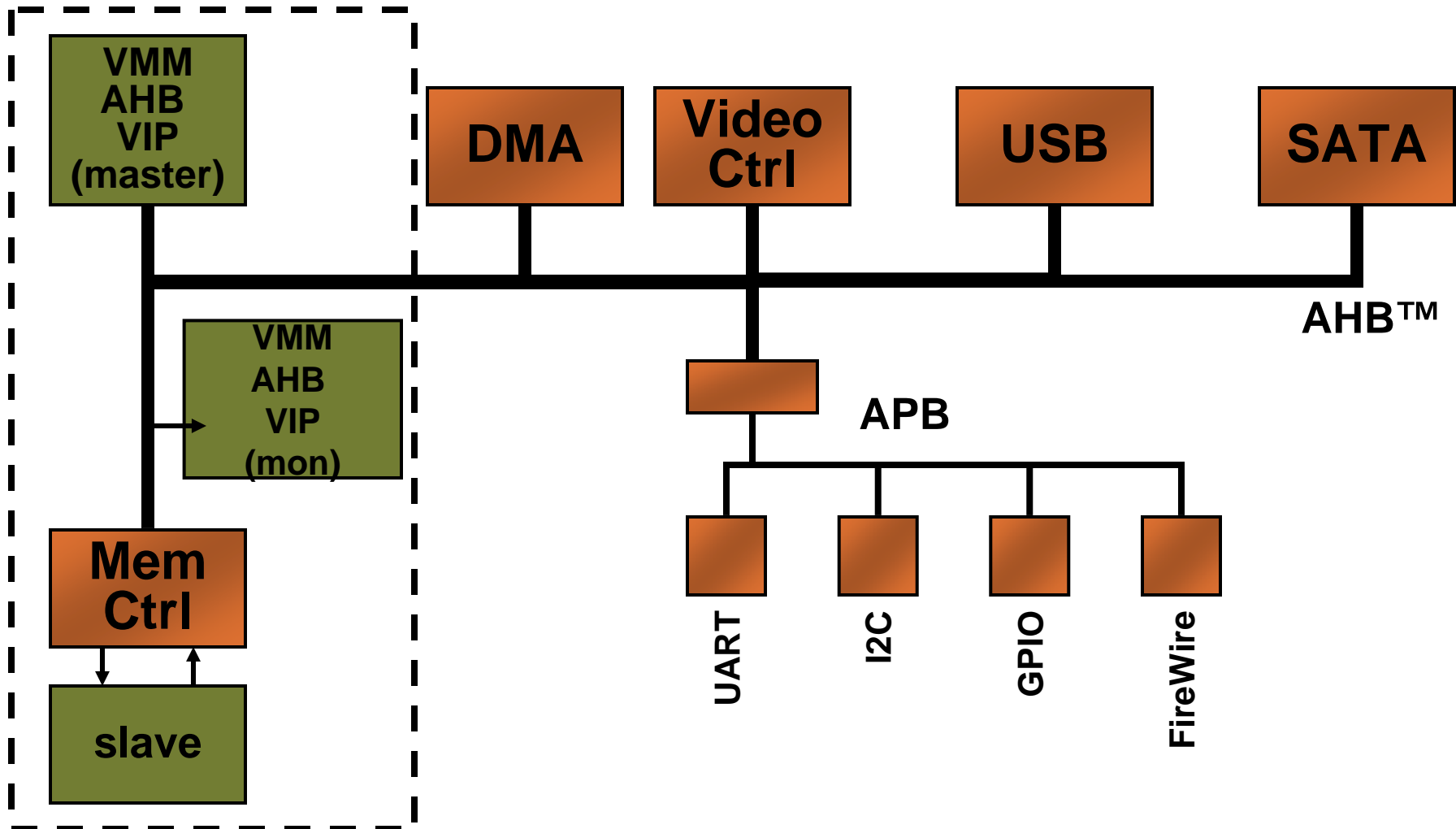
Many design environment, *Tools • IP • Services • Training*

- Highly configurable designs
- Multiple bus-based designs
- Multiple shared resources, memory based designs
  
- Networking and communication SOC and system development
  - Routers, switches
- Processors, DSP development
- Cache controllers
- Multi-core development
- Multi-threaded design development



# Bus controller verification

## AHB based Memory Controller Example



# Processor verification: (1)

## VLIW Processor based environment example

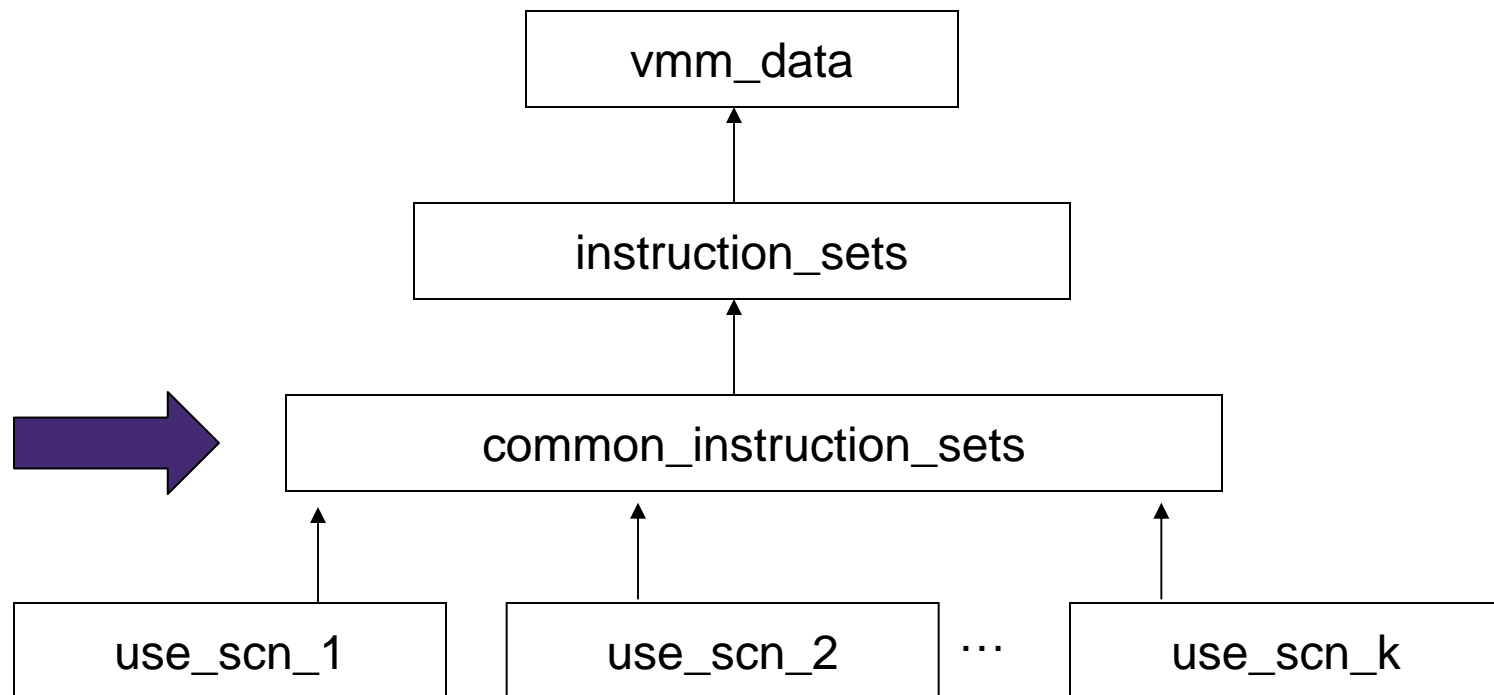
- Instruction: two slots
  - Format: 

Operation 0	;	Operation 1
-------------	---	-------------
  - Rules:
    - Slot-0 only operations (Load/Store, Return from Exception)
    - Memory address alignments (word aligned, i.e. 4 LSB=0)
    - \* Illegal Instruction sequence (no back to back branches)
    - \* Write-after-write Hazards (add r1, r2, r3 ; add r1, r3, r4)
- Other Features:
  - Exceptions
    - Arithmetic overflow, illegal instruction, misaligned address, watchpoints
    - Multiple exceptions possible and are resolved according to priority

## VMM base class usage (2)

### instruction set extension

- Use vmm\_data to extend common scenario base class



# Summary

- VMM promotes reuse and design quality
- Proven on many projects, in many design formats
- Harnesses full power of SystemVerilog for higher verification productivity
- Growing VMM user base and talent pool
- 3<sup>rd</sup> party vendors and consultants