2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

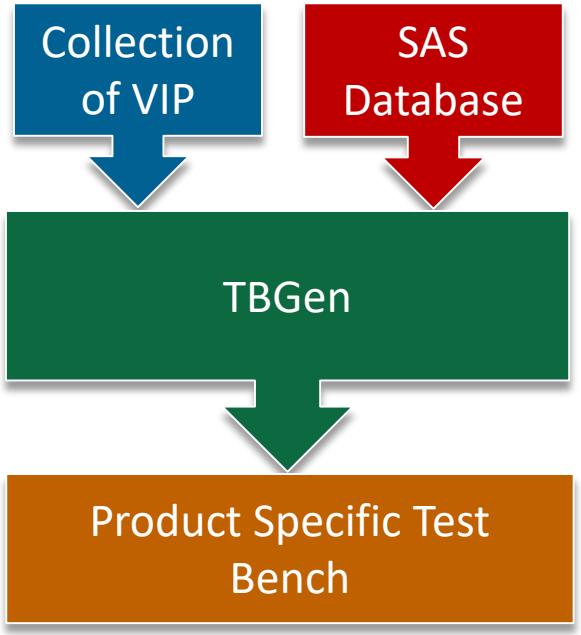
Automatic SOC Test Bench Creation

David Crutchfield, Cypress Semiconductor
Mark Glasser[1], NVIDIA Corporation
Stephen Roe, Cypress Semiconductor

[1] Contribution was given while at Cypress.

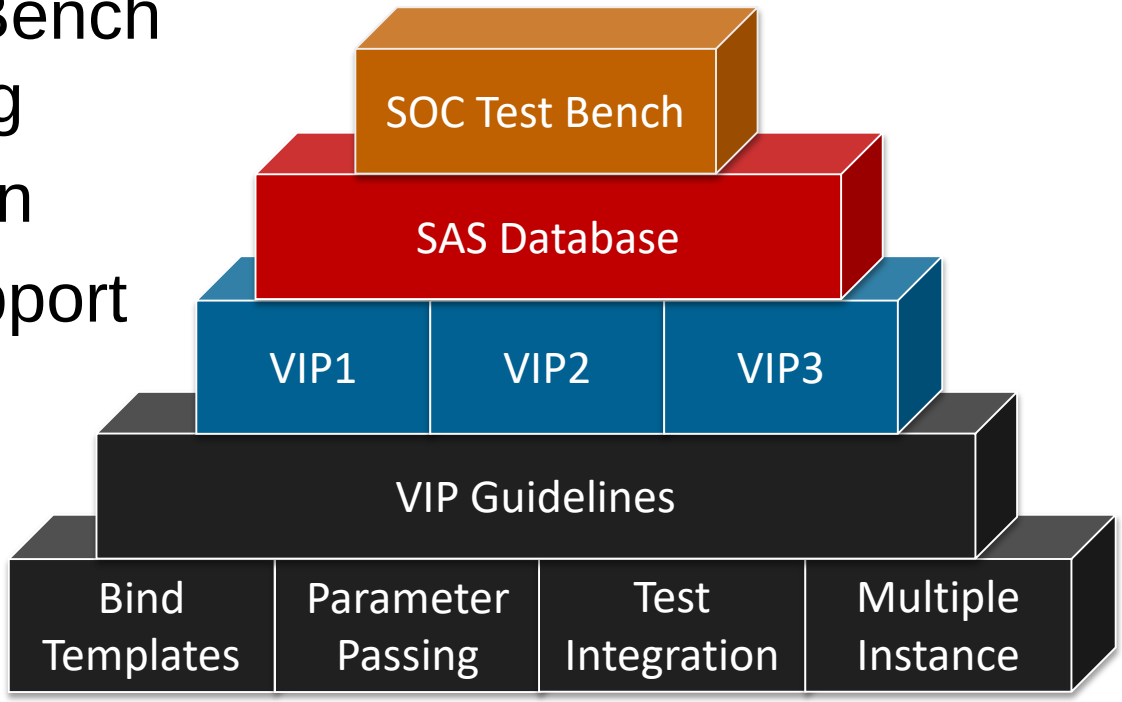
What is TBGen?

- Test Bench Generator
- Data driven system
- System Architecture Specification (SAS)
- Sister script to RTL Generator
- Integrates Verification IP (VIP) in a uniform way
- Little knowledge of IP / VIP



Building Blocks

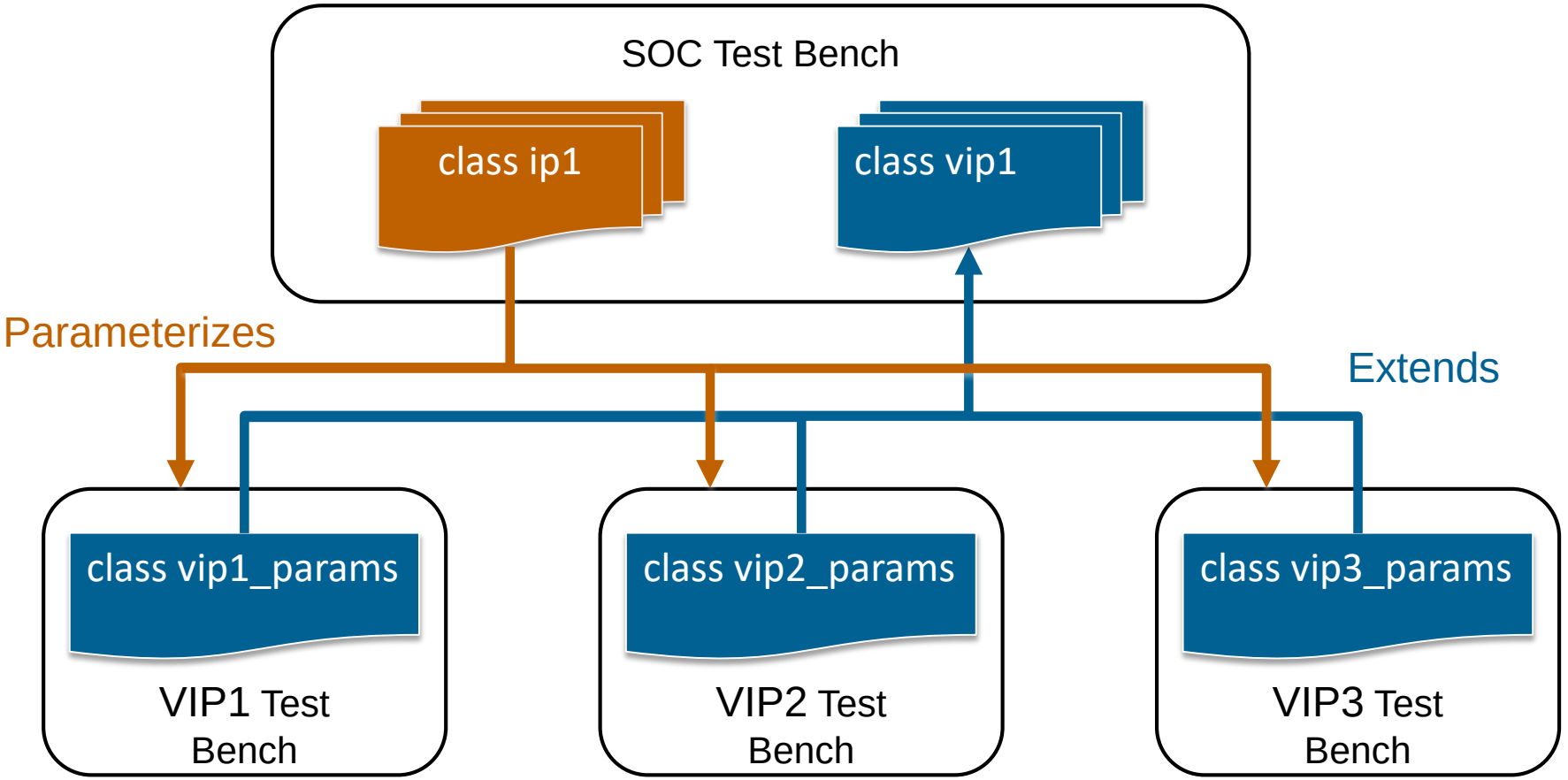
- Build VIP in a uniform way
- Uniform VIP Guidelines
  - Creation of interfaces through Bind Templates
  - Design and Test Bench parameter passing
  - VIP test integration
  - Multi-instance support
- SAS Database



Parameter Passing

- Configurable IP requires parameters.
- SAS provides design parameters.
  - Pieces of information used to configure design.
- Test bench uses design parameters to create test bench parameter class.
  - Test bench core is configured based on design and test bench parameter classes.
- Package sys
  - Contains both class types
  - Imported to gain access to both sets.

Flow of Parameters



Parameter Example

sys.sv

```
package sys;
   virtual class ip1;
      parameter PARAM1 = 1;
      parameter PARAM2 = 15;
   endclass

   `include “vsys.svh”
endpackage
```

sample.sv

```
import sys::*;

module sample;
   logic [ip1::PARAM2-1:0]          somebus;
   logic [vip1::TB_PARAM1-1:0]   tb_bus;
   ….
endmodule
```

vsys.svh

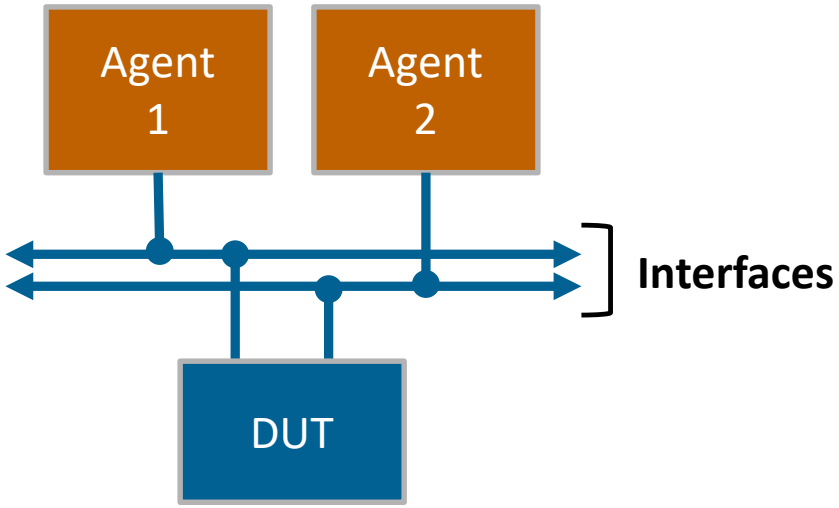
```
`include “vip1_params.svh”
class vip1 extends vip1_params;
endclass
```

vip1_params.svh

```
virtual class vip1_params;
   parameter TB_PARAM1 = 100;
endclass
```

Connections

- Generator must forge connections between test bench and RTL.
- Connections are made through interfaces.
- DUT interface
  - Collection of inputs / outputs to DUT.
- SystemVerilog interface
  - Means for representing wires as a single object.



Bind Templates

- Pieces of code identifying how to bind VIP to RTL.
- Used to generate bind-statements for interfaces and wrappers.
  - Also used to bind non-interfaces
    - Example: Assertion modules
- Contents
  - Unique bind options
  - Connection pairs

Bind Methodology

- Uniform naming
  - Field names derived from bind template name and contents.
  - Necessary for retrieval.
- Bind into subsystem level design scope.
- Interfaces and wrapper modules are self-contained.

Bind Options

- Bind template naming and parameters

Parameter	Options
bind_req	Design parameter expression(s).
generate / not_generate	Define label.
interface	No
	if_name, if_inst_name
	if_name, if_inst_array_name, if_limit
sas_param	Design parameter.
tb_param	Test bench parameter.

Example Bind Template

dut_wrapper.sv

```
module dut_wrapper;
`ifdef  IF_ON
   bind ip3: top.d.u_ip3_top
      sample_if #(
         .PARAM1(sys::ip3::PARAM1)
      )
      sample_if_0 (
         .if_sig1(input1),
         .if_sig2(input2)
      );
`endif
endmodule
```

vip3/sample_if-0

```
bind_req     = PARAM2 > 0
generate     = IF_ON
sas_param  = PARAM1

if_sig1      input1
if_sig2      input2
```

top.sv

```
module top;
   ovm_container#(virtual sample_if #(sys::ip3::PARAM1)::
      set_value_in_global_config(
         "ip3_sample_if_0", top.d.u_ip3_top.sample_if_0
      );
endmodule
```

Multi-instance Support

- Some IP need to be instanced more than once.
- Each instance may be configured differently.
  - With or without I2C.
  - Different number of SPI slaves.
  - Monitors for these need to be conditionally added.
- Design parameters control instance configuration.
- Design and test bench parameter classes unique to each instance must be passed to test bench.
  - Unique design class passed to v<ip>_params.svh.
  - Both design and test bench classes passed to env.

Multi-instance Example

SOC/env.svh

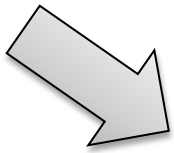
```
vip1_env#(sys::ip1_0, sys::vip1_0)
    m_ip1_0_env;

vip1_env#(sys::ip1_1,  sys::vip1_1)
    m_ip1_1_env;
```

SOC/sys.sv

```
virtual class ip1_0;
endclass

virtual class ip1_1;
endclass
```



VIP1/env.svh

```
class ip1_env#(type P, type V)
    extends ovm_env;

    ip1_sb#(P, V)    m_ip1_sb;
endclass
```

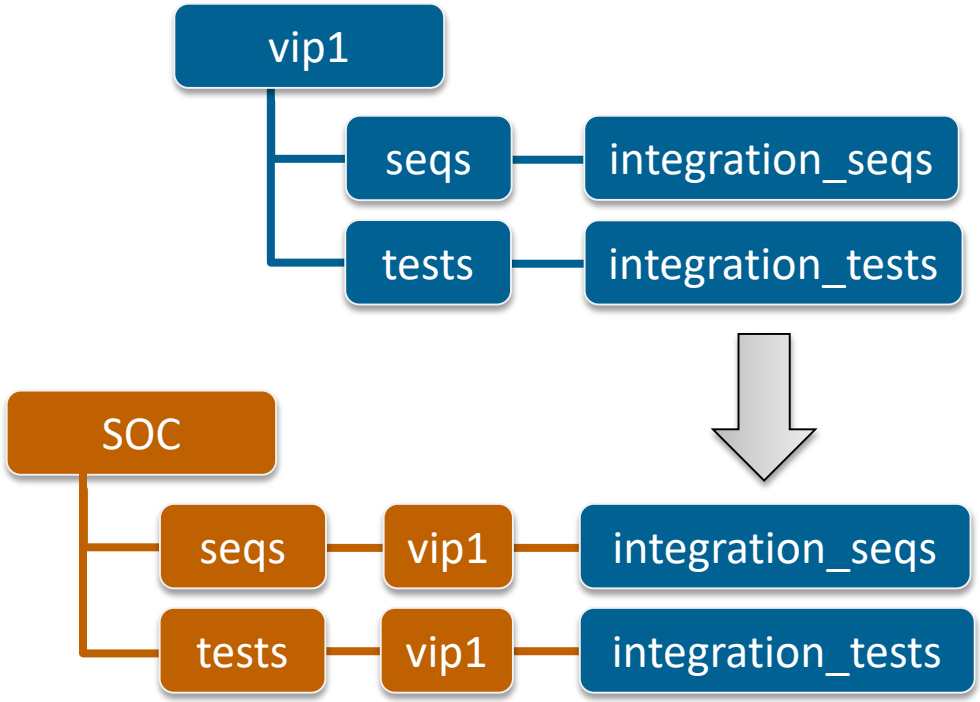
VIP1/vip1_params.svh

```
class vip1_params#(type P);
    parameter PARAM1 =
        5*P::PARAM1;
endclass
```

```
class ip1_sb#(type P, type V)
    extends ovm_env;
    localparam PARAM1 =
        P::PARAM1;
endclass
```

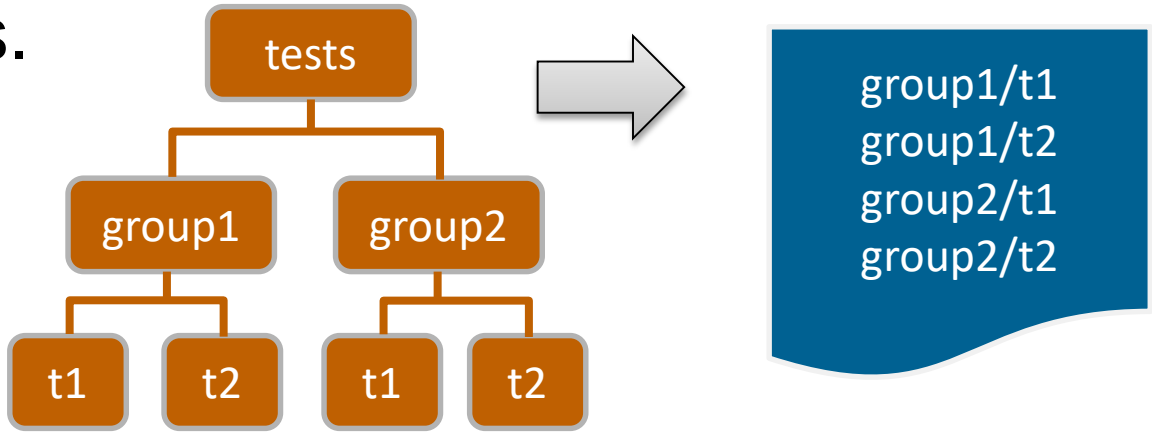
Test integration

- VIP contains integration tests in predefined location.
- TBGen simply copies integration sequences and tests into VSOC.
- TBGen automatically includes .svh files in packages.



Automatic Test List Creation

- Verification Management System (VMS)
- Gathers design and test bench files based on hierarchical file lists.
- Infers test list based on predefined hierarchical test tree structure.
- Launches tests.



Register Package Generation

- Mentor's HDL Designer Register Assistant.
- Input: Flattened register description files.
- Output: SystemVerilog register package.
- Flattened register description files are generated directly from the SAS.

Pin Access

- Pin mappings change based on design parameters.
- VIP will not know where pins are but still need access.
- Where needed programmable wrapper modules are bound into RTL for protocols like I2C, SPI, etc.
  - Pins are mapped based on SAS information.
  - Wrapper modules are configurable with pin parameter information.
  - VIP can use known wrapper module interfaces.
- Interfaces for each package provided.

Conclusion

- TBGen
  - Data driven system based on SAS
  - Uniform methodology
    - Parameters
    - Bind
    - Integration tests
    - Multi-instance
    - Register generation
- Push button test bench creation and integration test launch.