design&verification
conference&exhibition

February 21-23, 2007

Generating and Maintaining AVM-based Verification Environments

SystemVerilog FrameWorks™

by

Ambar Sarkar, Ph.D.
Chief Verification Technologist
Paradigm Works Inc.

PARADIGM® WORKS

Acknowledgements

- Mentor Graphics
- Paradigm Works
  - Charles Y. Choi, Ph.D.
  - Ning Guo, Ph.D.

Overview

- SystemVerilog FrameWorks™
- Router example
- AVM environment for router verification
- Testbench generation and maintenance
- Conclusion

SystemVerilog FrameWorks™

- A tool to jumpstart a SystemVerilog testbench environment
  - Reduces team ramp-up time significantly
  - Helps and enforces adherence to TLM/AVM/Other methodology
  - Captures verification expertise not easily accessible elsewhere
  - Promotes reuse and maintainability
- Proven on real projects across multiple clients

SystemVerilog FrameWorks™

- Scripts that automate many of the tasks
  - Creating templates
  - Creating directory structures
  - Compilation and execution
- Many base classes
  - Scoreboards, Test completion, Csrs etc
- Rich set of examples
  - Methodology compliant Verification components
  - Examples of unit/system-level tests
- Functional Coverage Driven Verification Plans
  - Template plans
  - Link with functional coverage data

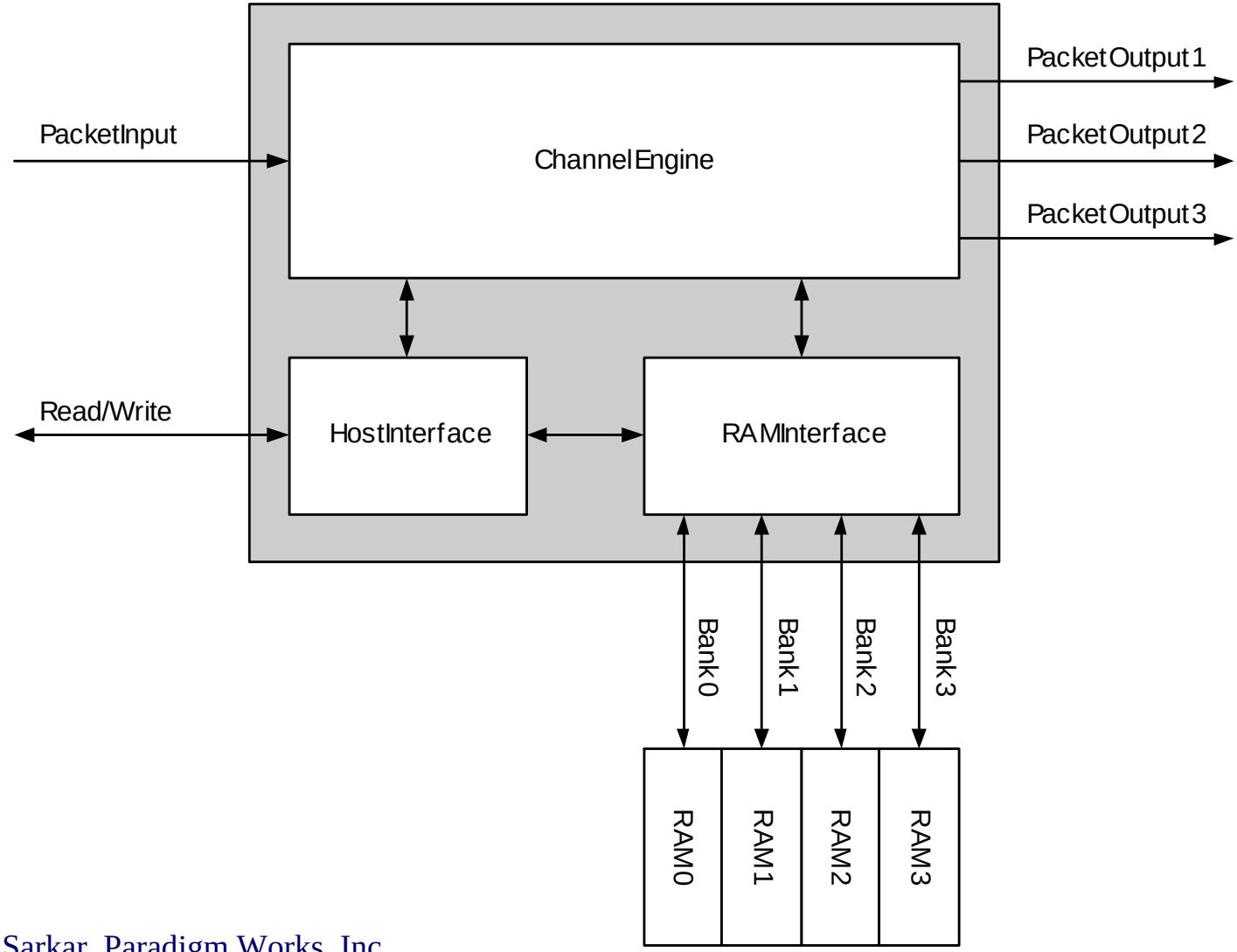
Components of SystemVerilog FrameWorks™

- Example of a layered verification environment
- Verification environment template generators
- Verification IP template generators
- Build and run scripts
- Register tool
- Functional coverage driven verification plan template
- Functional coverage tool
- etc..

Example of a Layered Verification Environment

- Example of end-to-end verification flow
  - A realistic example of a DUT
  - An example verification plan created
  - All environment code available
  - Example Verification IP code provided
  - Sample tests provided

Router Example



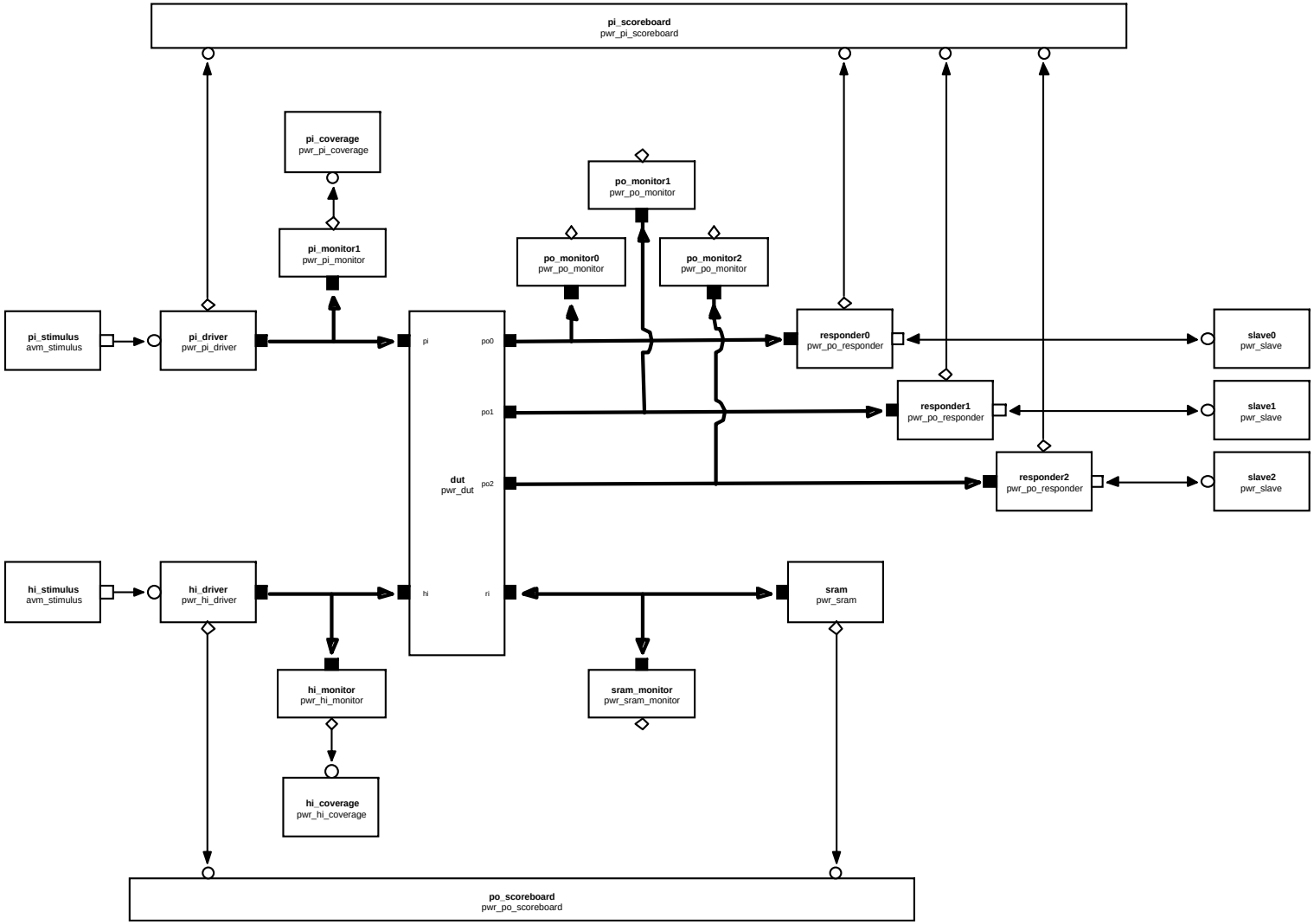
Router Example Requirements

- Stores and forwards packets to the output port
  - based on the address field of the packet
- Accesses each port through round-robin arbitration
- Implements a priority scheme
  - Allow high-priority packets through the device prior to any lower priority packets stored in memory for a single output port
- Supports off chip memory

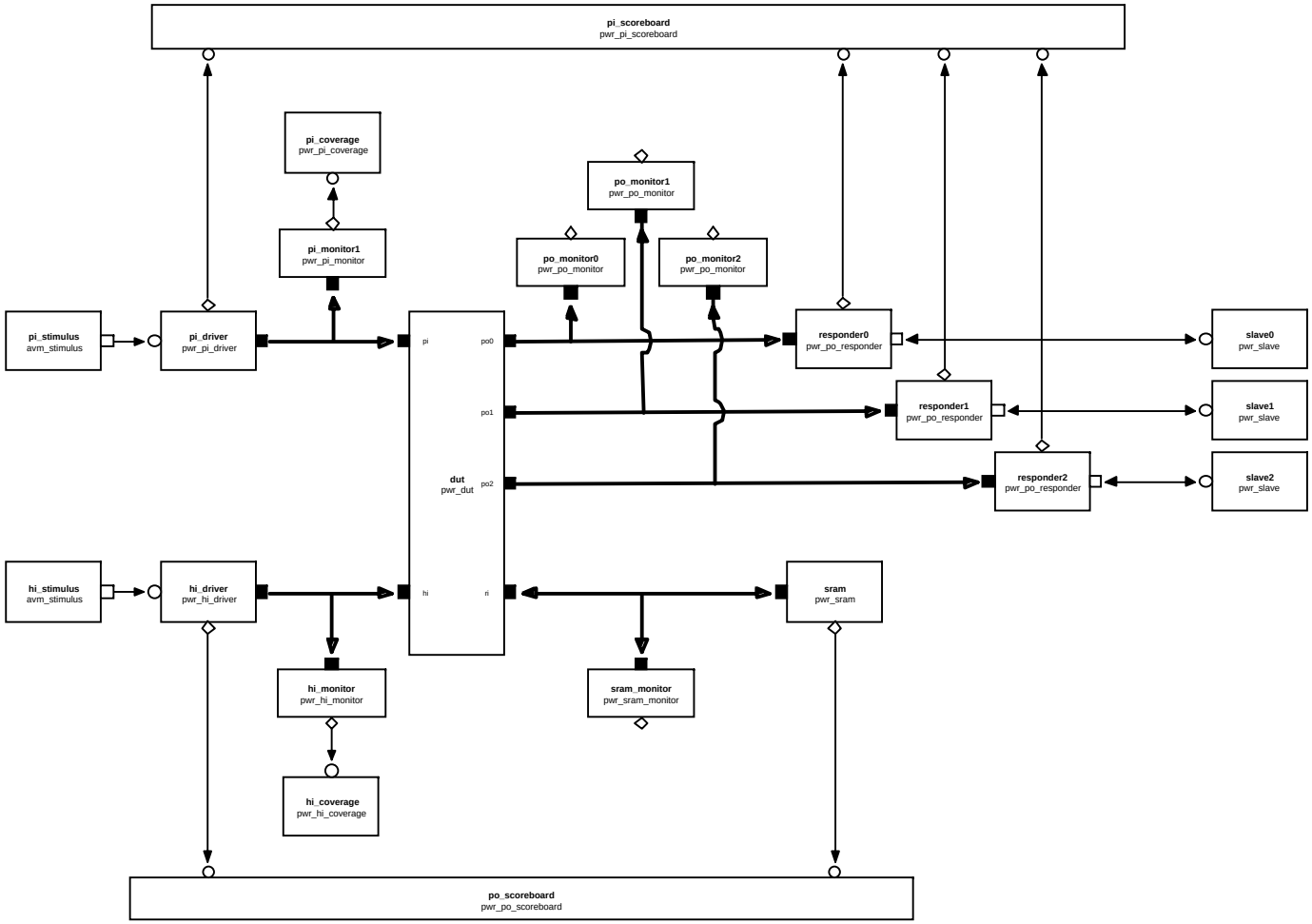
Router Example

- Forwards all packets to the output port
  - Indicates packets with bad parity
- Contains control and status registers
  - accessible by the host interface
- Allows CPU reads and writes to external memory
- The device has an initialization path including register operations
- Exception handling
  - Interrupts
  - Errors

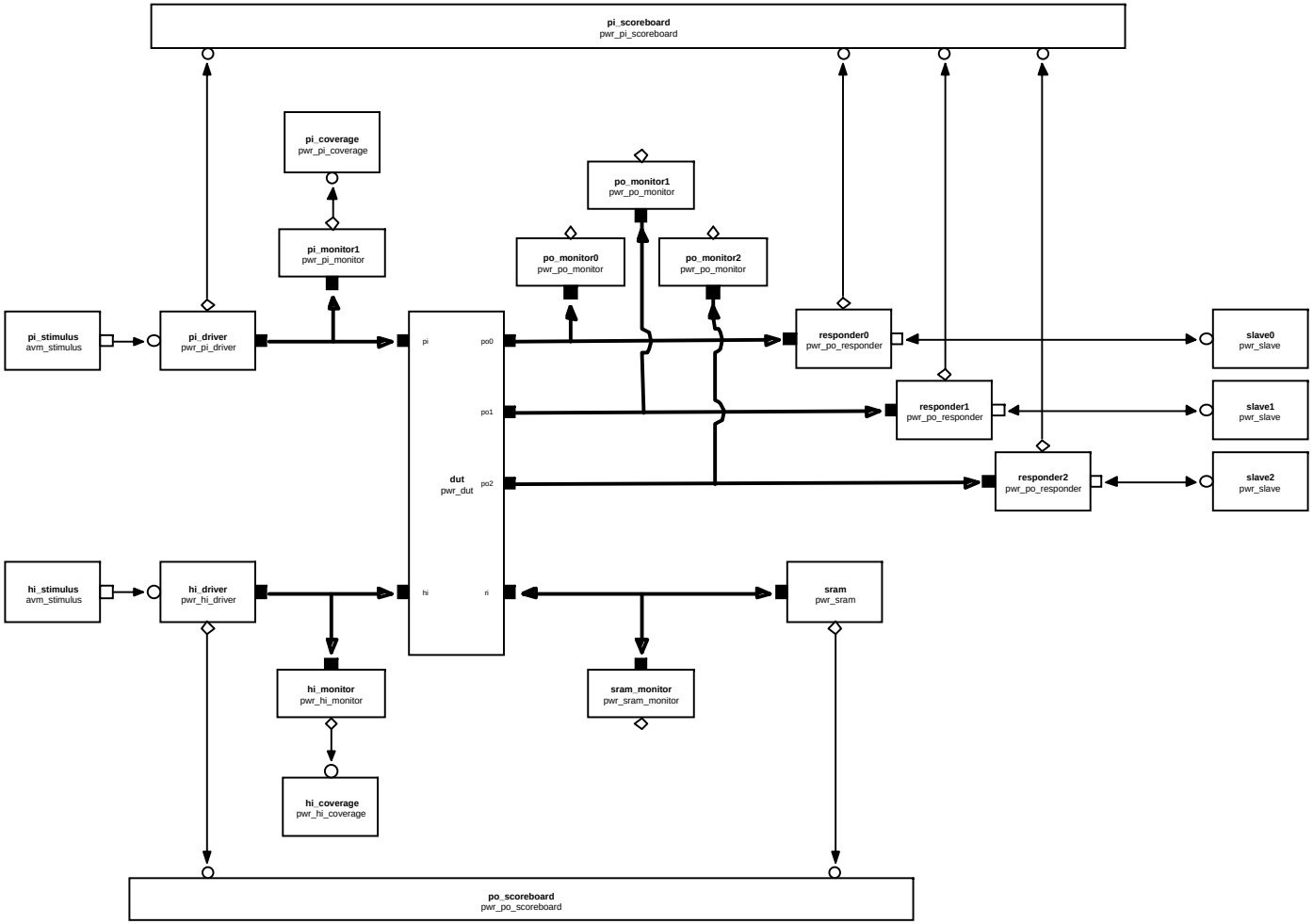
Router Verification Environment -Overall



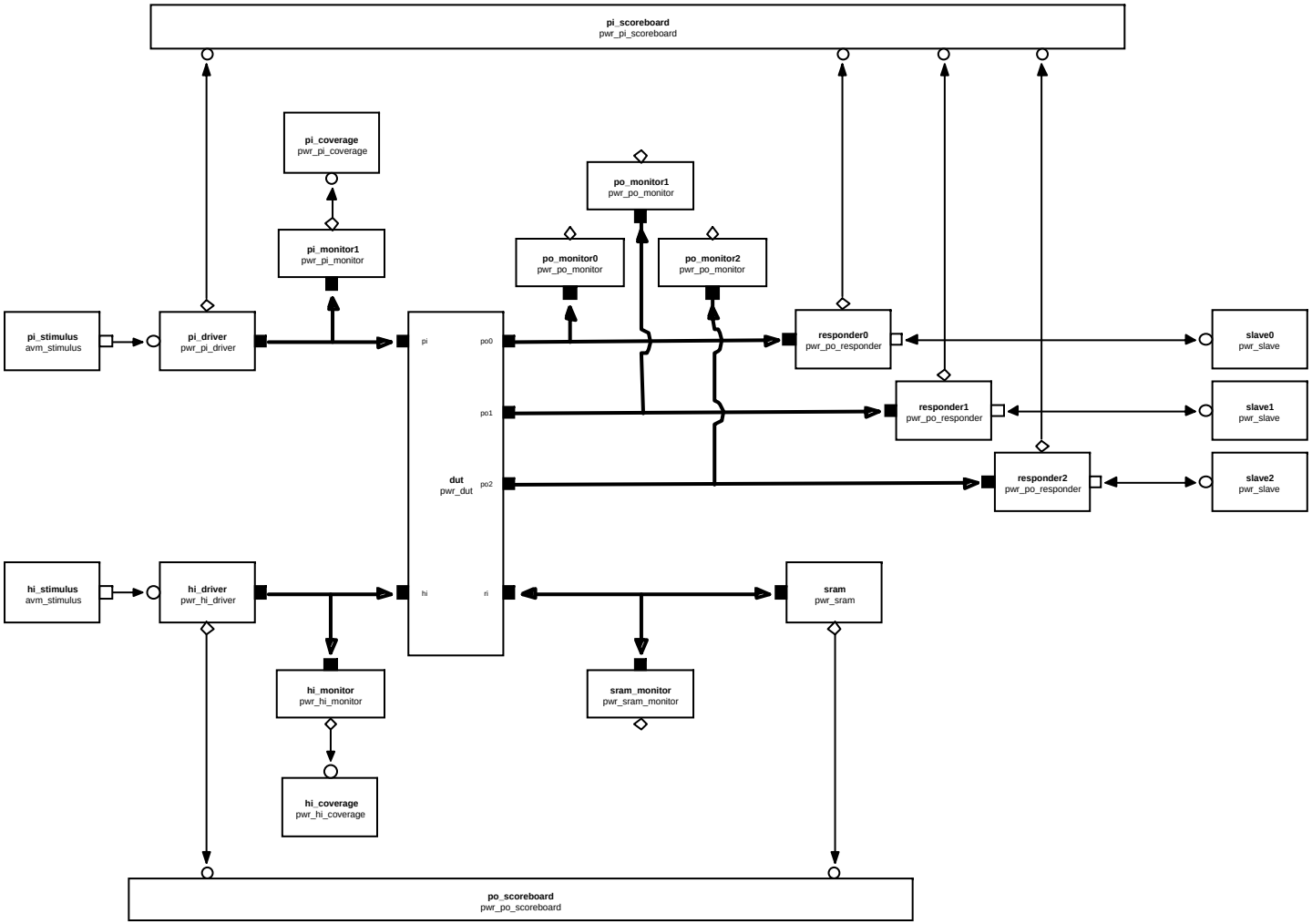
Router Verification Environment – Stimulus Generation



Router Verification Environment – Response Checking



Router Verification Environment – Functional Coverage



Router Verification Environment –Top Level

```
module top;
   ...
   pwr_pi_if
        pi_bus(  .clock(clock_reset_bus.clk),
                 .reset_n(clock_reset_bus.rst));
   pwr_po_if
        po0_bus( .clock(clock_reset_bus.clk),
                 .reset_n(clock_reset_bus.rst));
   pwr_po_if
        po1_bus( .clock(clock_reset_bus.clk),
                 .reset_n(clock_reset_bus.rst));
   pwr_po_if
        po2_bus( .clock(clock_reset_bus.clk),
                 .reset_n(clock_reset_bus.rst));
   pwr_hi_if
        hi_bus( ...
   ...
   pwr_env env;

   initial  begin
       env = new(pi_bus, po0_bus,po1_bus,po2_bus,hi_bus);
   ...
```

Router Verification Environment –Top Level contd..

```
router dut(
   .clock(clock_reset_bus.clk),
         .reset_n(clock_reset_bus.rst),
         .packet_input_data        (pi_bus.packet_input_data),
         .packet_input_valid       (pi_bus.packet_input_valid),
         .packet_input_suspend     (pi_bus.packet_input_suspend),

         .packet_out_data_0        (po0_bus.packet_out_data),
         .packet_out_valid_0       (po0_bus.packet_out_valid),
        ...
         .hif_address              (hi_bus.target_mp.hif_address),
         .hif_data                 (hi_bus.target_mp.hif_data),
        ...
        .interrupt                 (hi_bus.target_mp.interrupt),
```

Router Verification Environment –Env

```
package pwr_env_pkg;
   import avm_pkg::*;
   import pwr_pkg::*;

   `include "pwr_report_formatter.svh"

   class pwr_env extends avm_env;

         pwr_pi_stimulus #(pwr_pi_transaction) pi_stimulus;
         pwr_pi_driver pi_driver;
         pwr_pi_monitor pi_monitor;
         pwr_pi_coverage #(pwr_pi_transaction) pi_coverage;
         pwr_pi_scoreboard #(pwr_pi_transaction) pi_scoreboard;

         pwr_po_responder responder[3];
         pwr_slave slave[3];
         pwr_po_monitor po_monitor[3];

         virtual pwr_pi_if pi_bus;
         virtual pwr_po_if po_bus[3];

         pwr_report_formatter report_formatter;
         ...
```

Router Verification Environment –Env contd.

```
function new( virtual pwr_pi_if pi_bus,
      virtual pwr_po_if po0_bus, virtual pwr_po_if po1_bus,
      virtual pwr_po_if po2_bus);
...
      this.pi_bus = pi_bus;
      this.po_bus[0] = po0_bus; this.po_bus[1] = po1_bus; ...

      this.pi_stimulus = new("pi_stimulus");
      this.pi_driver = new("pi_driver");

      for (int i=0; i < 3; i++) begin
          $sformat(s, "%0d", i);
          this.responder[i] = new({"responder", s});
          this.slave[i] = new({"slave", s});
          this.po_monitor[i] = new({"po_monitor", s});
      end

      this.pi_monitor = new("pi_monitor");
      this.pi_coverage = new("pi_coverage");
      this.pi_scoreboard = new("pi_scoreboard");

      this.report_formatter = new(); this.report_formatter.longformat = 0;
      set_report_formatter(this.report_formatter);

      ...
endfunction // new
```

Router Verification Environment –Env contd.

```
function void connect();
   this.pi_stimulus.blocking_put_port =
                          this.pi_driver.blocking_put_export;
   this.pi_driver.ap.register(this.pi_scoreboard.add_before_input());

   this.pi_driver.bus_if = this.pi_bus;
   this.pi_monitor.bus_if = this.pi_bus;

   this.pi_monitor.ap.register(this.pi_coverage.analysis_export);


   for (int i=0; i < 3; i++) begin
         this.responder[i].bus_if = this.po_bus[i];
         this.slave[i].blocking_get_export =
         this.responder[i].blocking_get_export;
         this.po_monitor[i].bus_if = this.po_bus[i];
         this.responder[i].ap.register(
                  this.pi_scoreboard.add_after_input());
   end

endfunction:connect
```

Router Verification Environment –Env contd.

```
   task execute();
      this.pi_driver.reset_dut();
      for (int i=0; i < 3; i++)
         this.po_monitor[i].reset_dut();
      fork
       begin
         this.stimulus_process = process::self;
         this.pi_stimulus.generate_stimulus();
       end
       this.terminate_when_covered();
      join
   endtask:execute

   task terminate_when_covered();
      wait(this.pi_coverage.is_covered);
      this.stimulus_process.kill();
      avm_report_warning("terminate_when_covered", "");
   endtask:terminate_when_covered

   endclass:pwr_env

endpackage:pwr_env_pkg
```

Router Verification Environment –VIP

```
class pwr_pi_driver extends avm_verification_component;

  virtual pwr_pi_if bus_if;
  ...
  tlm_blocking_get_if #(pwr_pi_transaction) blocking_get_export;
  tlm_blocking_put_if #(pwr_pi_transaction) blocking_put_export;
  local tlm_fifo #(pwr_pi_transaction) fifo;
  analysis_port #(pwr_pi_transaction) ap;

  function void export_connections();
     this.blocking_get_export = this.fifo.blocking_get_export;
     this.blocking_put_export = this.fifo.blocking_put_export;
  endfunction:export_connections

  function new(string name, avm_named_component parent=null);
     super.new(name, parent);
     this.fifo = new("pi_fifo", this); this.ap = new();
  endfunction:new

  task run();
     this.transmit_t();
  endtask:run

  task transmit_t();
   ...
  endtask:transmit_t
endclass:pwr_pi_driver
```

Router Verification Environment – Directory Structure

```
./common                                    # Common files
./common/common.list                        # Build list common files
./common/pwr_report_formatter.svh           # Common class

./vips                                      # Dir of all vips
./vips/src.list                             # Build file for all vips
./vips/pwr_pkg.sv                           # Top-level vip pkg

./vips/pwr_hi                               # Directory for pwr_hi
./vips/pwr_hi/doc                           # Documentation for pwr_hi
./vips/pwr_hi/src.list                      # Build file for pwr_hi
./vips/pwr_hi/pwr_pi_pkg.sv                 # Top-level pkg for pwr_hi
./vips/pwr_hi/pwr_hi_if.sv                  # Interface for pwr_hi
./vips/pwr_hi/pwr_hi_transaction.svh        # Transaction for pwr_hi
./vips/pwr_hi/pwr_hi_stimulus.svh           # Stimulus for pwr_hi
./vips/pwr_hi/pwr_hi_driver.svh             # Driver for pwr_hi
./vips/pwr_hi/pwr_hi_monitor.svh            # Monitor for pwr_hi
./vips/pwr_hi/pwr_hi_responder.svh          # Responder for pwr_hi
./vips/pwr_hi/pwr_hi_slave.svh              # Slave for pwr_hi
./vips/pwr_hi/pwr_hi_coverage.svh           # Coverage for pwr_hi
./vips/pwr_hi/pwr_hi_scoreboard.svh         # Scoreboard for pwr_hi
```

Router Verification Environment- Directory Structure

```
./vips/pwr_pi                                  # Directory for pwr_pi
./vips/pwr_pi/doc                              # Documentation for pwr_pi
./vips/pwr_pi/src.list                         # Build file for pwr_pi
./vips/pwr_pi/pwr_pi_pkg.sv                    # Top-level pkg for pwr_pi
./vips/pwr_pi/pwr_pi_if.sv                     # Interface for pwr_pi
./vips/pwr_pi/pwr_pi_transaction.svh           # Transaction for pwr_pi
./vips/pwr_pi/pwr_pi_stimulus.svh              # Stimulus for pwr_pi
...
./sys                                          # Environment
./env/env.list                                 # Build file for environment
./env/pwr_env_pkg.sv                           # Pkg for top level
./env/top.sv                                   # Top-level

./test                                         # Test directory
./test/src.list                                # Src list

./test/pwr_hi_test                             # Tests for pwr_hi
./test/pwr_hi_test/hi_test1.svh
./test/pwr_pi_test                             # Tests for pwr_pi
./test/pwr_pi_test/pi_test1.svh

./doc                                          # Overall documentation
```

Challenges of generating AVM environments

- Overcoming initial learning curve
- Working with teams with mixed skill-levels
- Following company specific requirements
- Supporting common practice
- Avoiding evils of cut and paste
- Communicating guidelines and conventions across multiple sites
- Capturing changes and applying lessons learned from previous projects

Automatic Testbench Generation: SVF-TG

```
Description:
   The SVF-TG tool takes as input a set of templates and will
   generate a SystemVerilog AVM test environment.
Options:
  -env                    -env env1 -env sys
  -help                   Print this help
  -installDir             Target installation directory
  -newttdir               Directory of the new template directory
  -projName               Name of project
  -testGroup              -te "<env name> <group name> <testName>
   <testName>"
                          <envName> = #any# means all environments
                          have these tests
  -ttdir                  Template directory
  -version                Print version information and exit.
  -vip                    -vip vipA -vip vip2[n] ...
```

Automatic Testbench Generation: SVF-TG

```
svfTG                              # Name of the executable
   -p pwr                          # Name of project
   -vip pwr_pi                     # pwr_pi vip, one instance
   -vip pwr_hi                     # pwr_hi vip, one instance
   -vip “pwr_po[3]”                # pwr_po vip, three instances
   -vip “ram[4]”                   # memory vip, four instances
   -env sys                        # sys environment
   -te “sys pwr_pi pi_test1 " # sys environment pwr_pi test
   -te “sys pwr_hi hi_test1 " # sys environment pwr_hi test
```

Automatic Testbench Generation: SVF-TG

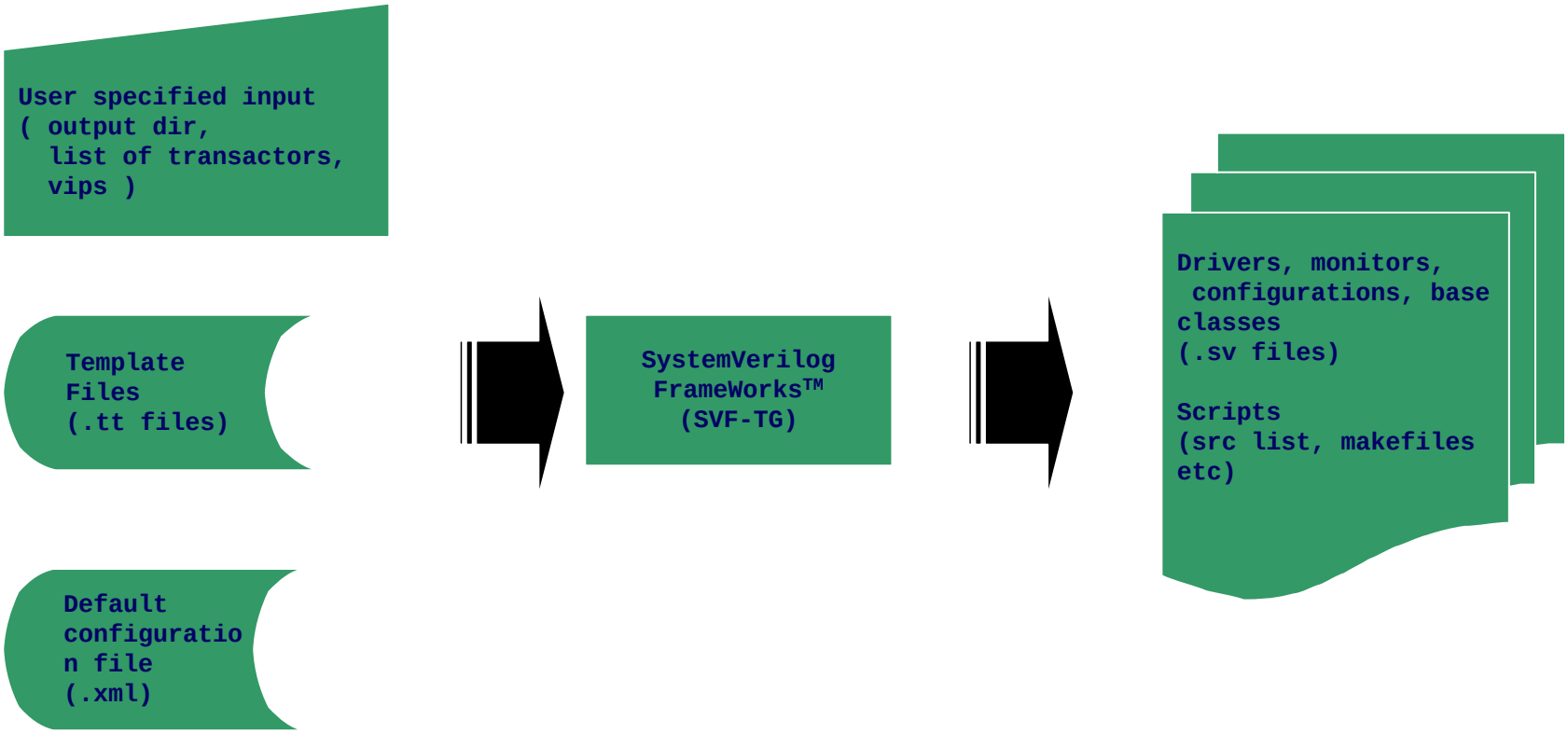
```
//-----------------------------------------------------------------------------
//
//                    Confidential and Proprietary Information
//
// The information and descriptions contained herein embody confidential and
// proprietary information which is the property of Paradigm Works, Inc. Such
// information and descriptions may not be copied, reproduced, disclosed to
// others, published or used, in whole or in part, for any purpose other than
// that for which it is being made available without the express prior written
// permission of Paradigm Works, Inc.
//
// Paradigm Works, Inc.,                    Copyright (c) 2001-2007
//-------------------------------------------------------------------------------
/*************************************************************************
 * Author:                    $Author:$
 * File:                      $File:$
 * Revision:                  $Revision:$
 * Date:                      $Date:$
 ***************************************************************************
 * Interface file pwr_pi_if.sv for vip pwr_hi in pwr verification environment
 ****************************************************************************/
`ifndef PWR_PI_IF
`define PWR_PI_IF
     interface pcie_if(input wire clk);
               logic [15:0] rdata;
               logic [15:0] wdata;
               logic write;

               // Add your own signals

                // For the DUT
               modport DUT( input wdata, output rdata, input write);

               // For the testbench
                modport TB( input rdata, output wdata, output write);
     endinterface: pwr_pi_if
`endif //PWR_PI_IF
```

Automatic Testbench Generation: SVF-TG



Automatic Testbench Generation: SVF-TG

```
...
/*********************************************************************
 * Interface file [% fileName %] for vip [% vip %] in [% projName %]
   verification environment
 *********************************************************************/
`ifndef [% projName FILTER upper %]_[% vip FILTER upper %]_IF
`define [% projName FILTER upper %]_[% vip FILTER upper %]_IF

interface [% vip %]_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals

   // For the DUT
   modport DUT( input wdata, output rdata, input write);
   // For the testbench
   modport TB( input rdata, output wdata, output write);
endinterface: [% vip %]_if

`endif //[% projName FILTER upper %]_[% vip FILTER upper %]_IF
```

Testbench Maintenance – Original Template

```
/*****************************************************************************
   *
 * Interface file [% fileName %] for vip [% vip %] in [% projName %]
   verification environment

   ****************************************************************************
   ****/
`ifndef [% projName FILTER upper %]_[% vip FILTER upper %]_IF
`define [% projName FILTER upper %]_[% vip FILTER upper %]_IF

interface [% vip %]_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals

   // For the DUT
   modport DUT( input wdata, output rdata, input write);
   // For the testbench
   modport TB( input rdata, output wdata, output write);
endinterface: [% vip %]_if
`endif //[% projName FILTER upper %]_[% vip FILTER upper %]_IF
```

Testbench Maintenance – New Template

```
`ifndef [% projName FILTER upper %]_[% vip FILTER upper %]_IF
`define [% projName FILTER upper %]_[% vip FILTER upper %]_IF

interface [% vip %]_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals

   // For the DUT
   modport DUT( input wdata, output rdata, input write);
   // For the testbench
   modport TBDRV( input rdata, output wdata, output write);
   modport TBMON( input rdata, input wdata, inut write);
endinterface: [% vip %]_if
```

Testbench Maintenance –Original code

```
interface pcie_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals

   // For the DUT
   modport DUT( input wdata, output rdata, input write);
   // For the testbench
   modport TB( input rdata, output wdata, output write);
endinterface: pcie_if
```

Testbench Maintenance-Modified Original

```
interface pcie_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals
   logic err;

   // For the DUT
   modport DUT( input wdata, output rdata, input write, output err);
   // For the testbench
   modport TB( input rdata, output wdata, output write, input err);
endinterface: pcie_if
```

Testbench Maintenance-Merged

```
interface pcie_if(input wire clk);
   logic [15:0] rdata;
   logic [15:0] wdata;
   logic write;
   // Add your own signals
   logic err;

   // For the DUT
   modport DUT( input wdata, output rdata, input write, output err);
   // For the testbench
<<<<<<< CURRENT
   modport TB( input rdata, output wdata, output write, input err);
||||||| OLDTMPL
   modport TB( input rdata, output wdata, output write);
=======
   modport TBDRV( input rdata, output wdata, output write);
   modport TBMON( input rdata, input wdata, input write);
>>>>>>> NEWTMPL
endinterface: pcie_if
```

Conclusion

- Creating and maintaining verification environments across multiple sites and groups is challenging.
- The SVF-TG tool allows teams to capture the project specific guidelines in an executable form so that complete and detailed verification environment shells can be customized and generated from scratch.
- In addition, the tool is able to handle changes in the guidelines during the life of a project by automatically merging newer code with pre-existing code and pointing out where there are conflicts that can only be resolved manually.
- AVM methodology helps significantly in maintaining verification environments