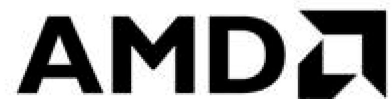
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

A Unified Testbench Architecture Solution for Verifying Variants of A PLL IP

Deepa Ananthanarayanan

Malathi Chikkanna



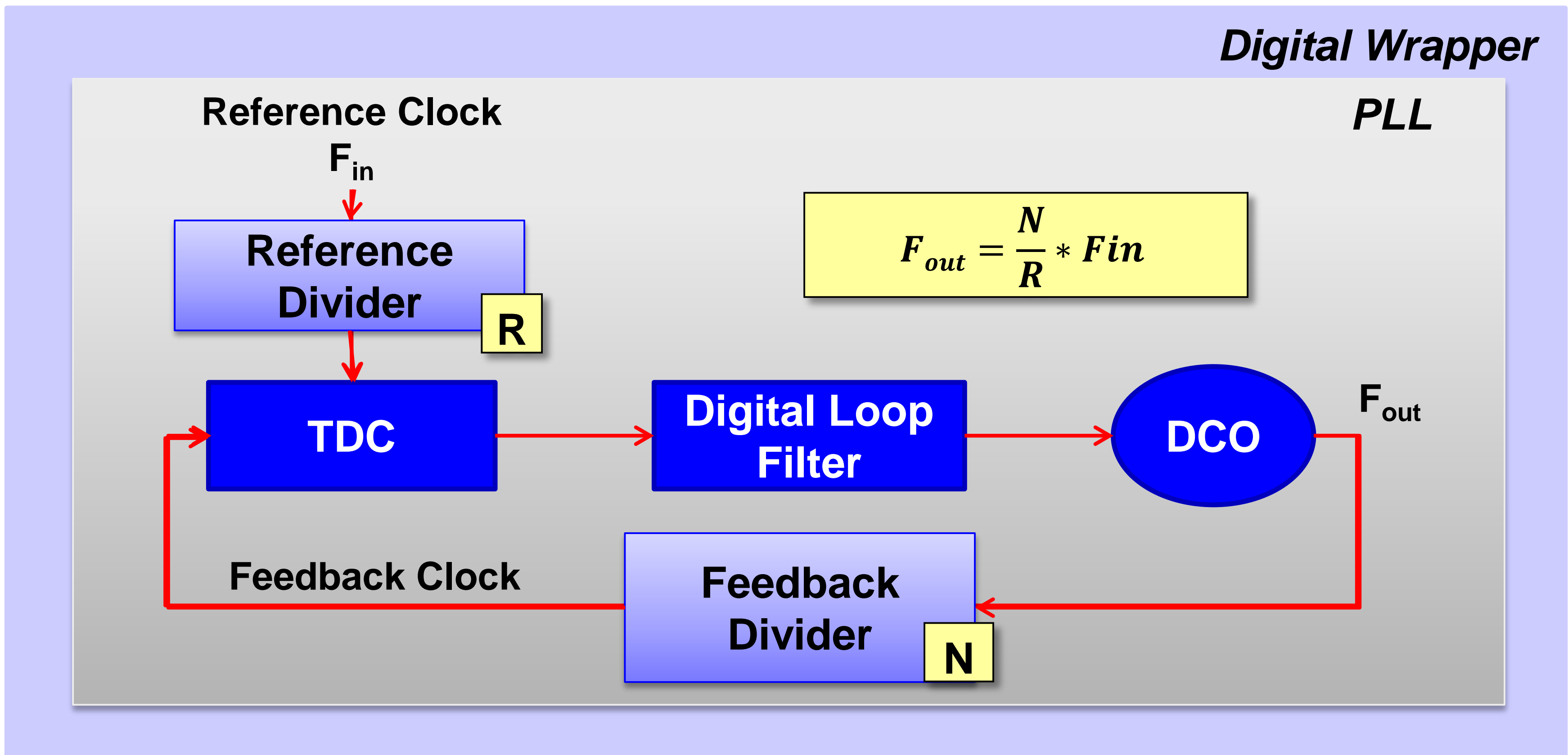
DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2015

AGENDA

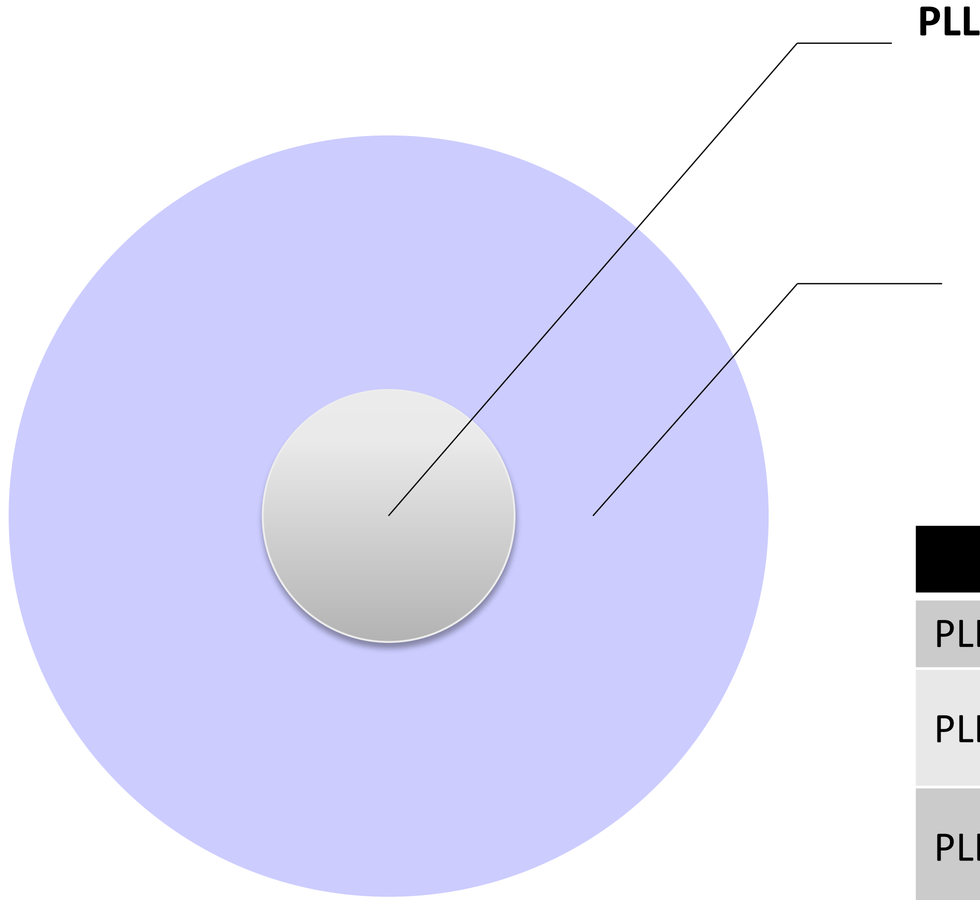
- Overview of PLL verification
- Verification Challenges in Prior work
- Proposed Solution
- Outcomes & Conclusion

DUT - Digital Phase Locked Loop IP

- A versatile clock synthesizer
- Used for Clock generation and distribution applications



A Typical PLL Variant



PLL

Digital Wrapper

- Reference Divider
- Post Divider
- Register
- Level Shifter
- DFT logic
- Custom design

IP	DUT
PLL	PLL 1
PLL + Digital Wrapper 2	PLL 2
PLL + Digital Wrapper 3	PLL 3
PLL + Digital Wrapper 4	PLL 4

PLL Verification Goal

Verify the functional correctness of the below features…

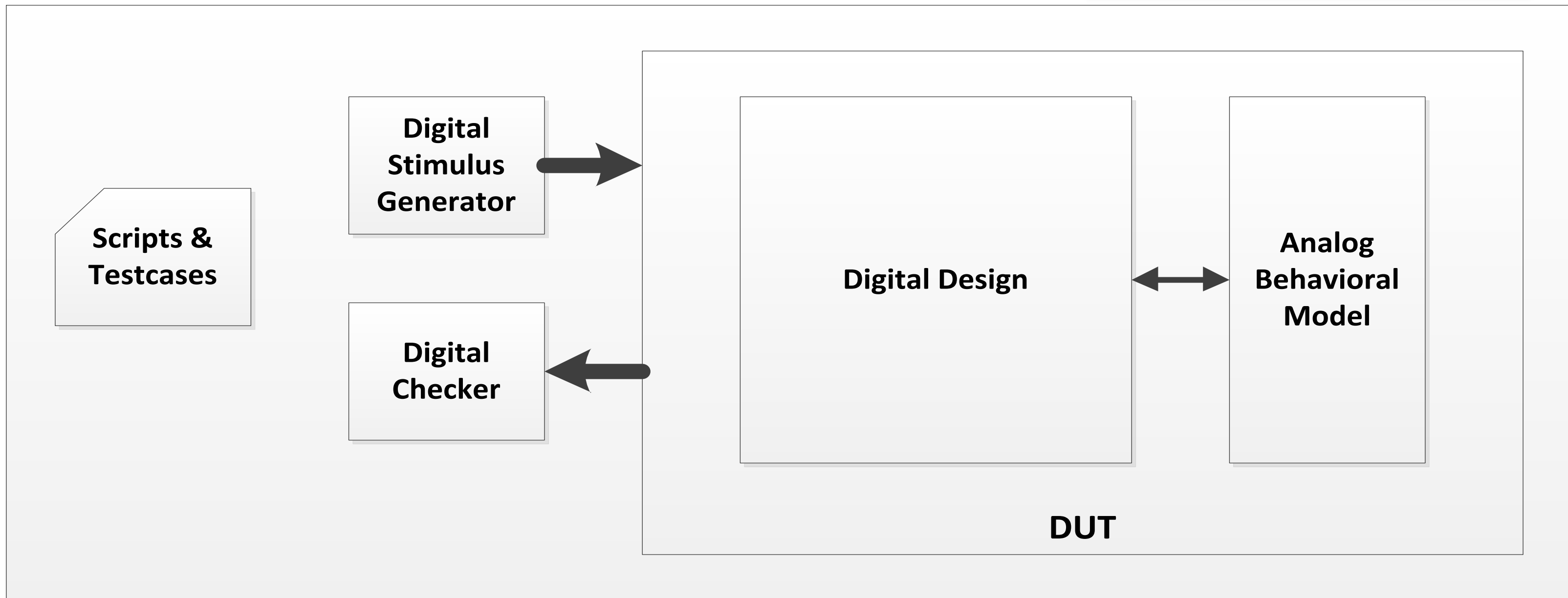
- Supported frequencies
- Custom Interface logic
- DFT
- Low Power
- etc…

…for 4 different variants of a PLL

IP	DUT
PLL	PLL 1
PLL + Digital Wrapper 1	PLL 2
PLL + Digital Wrapper 2	PLL 3
PLL + Digital Wrapper 3	PLL 4

How did we verify the DUTs?

- ✓Verilog based testbench (TB)
- ✓Unique testbench per PLL variant



AGENDA

- Overview of PLL verification
- Verification Challenges in Prior work
- Proposed Solution
- Outcomes & Conclusion

Challenges in the prior verification approach

Challenges

- Lack of constrained random verification
- Poor testbench re-usability
- Poor scalability
- Lack of Portability and
- Configurability

Potential solutions

- Adopting *SystemVerilog*
- Creating a common library of generic tasks and functions

Single Comprehensive PLL Testbench Architecture

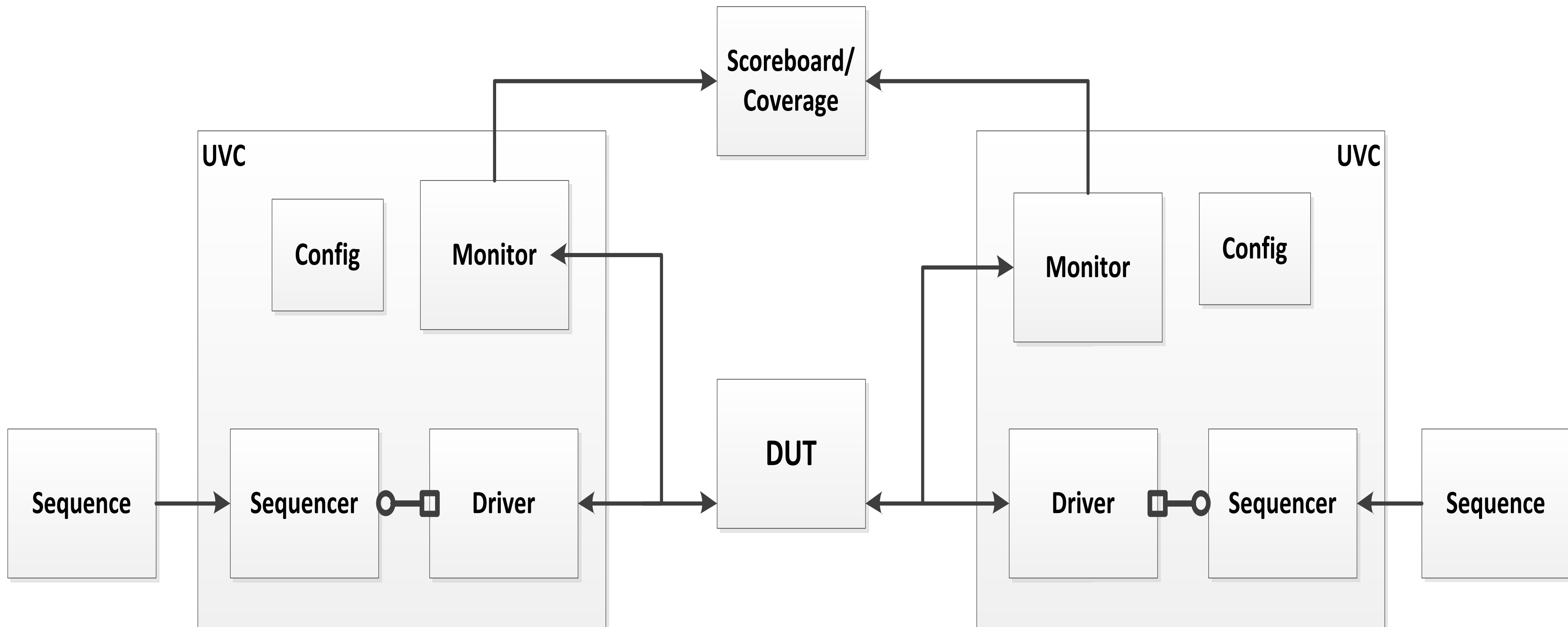
Adopting Universal Verification Methodology using SystemVerilog

AGENDA

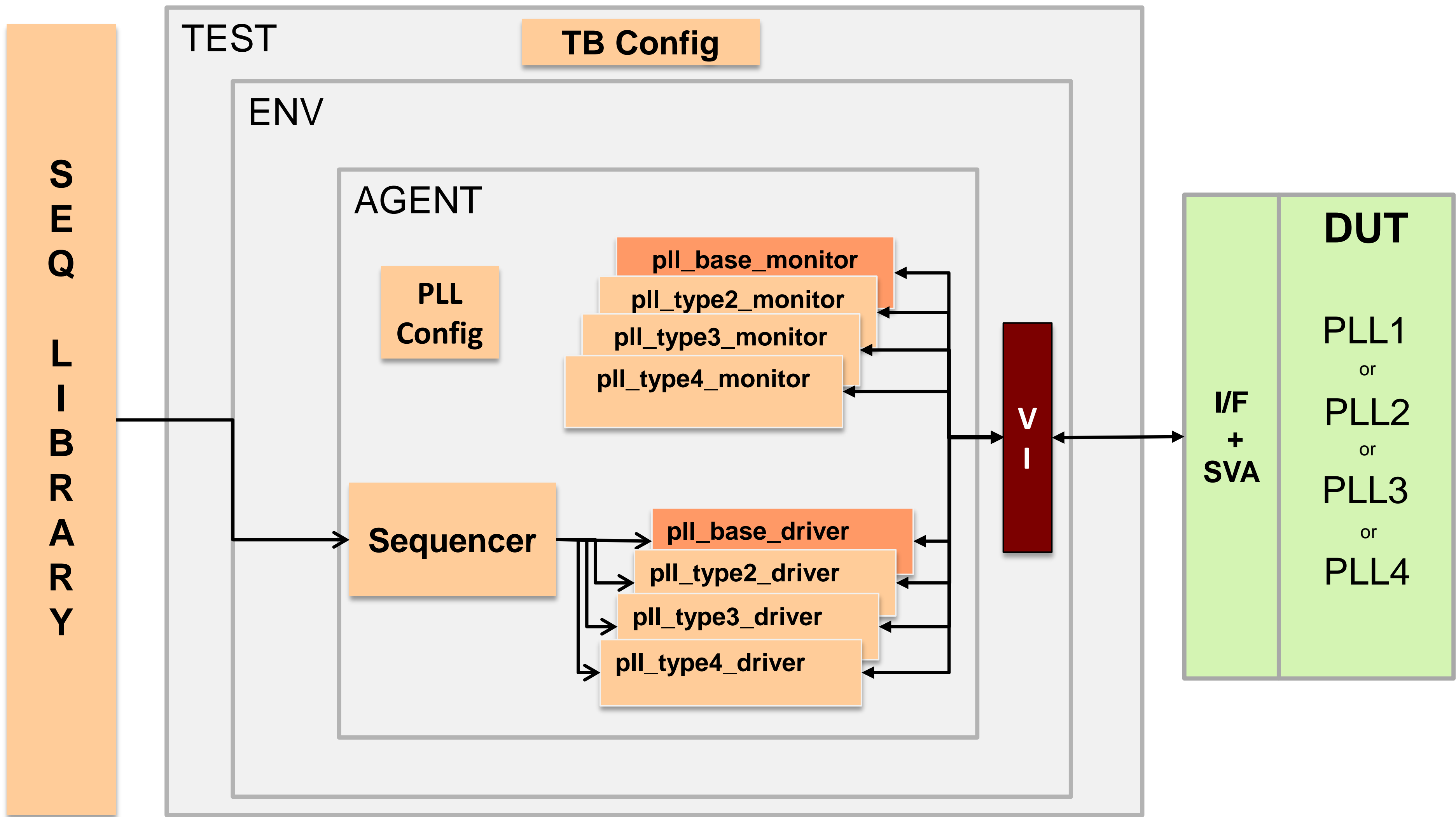
- Overview of PLL verification
- Verification Challenges in Prior work
- Proposed Solution
- Outcomes & Conclusion

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Typical UVM Testbench



Proposed Solution Unified TB Architecture



PLL AGENT

```
class pll_agent extends uvm_agent;

        pll_sequencer         mast_sqrh;
        pll_base_driver       mast_drvh;
        pll_base_monitor      mast_monh;

        pll_cfg               m_pll_cfg
….
endclass: pll_agent

function void pll_agent::build_phase(uvm_phase phase);
        super.build_phase(phase);

        mast_monh = pll_base_monitor::type_id::create("mast_monh", this);

        if (m_pll_cfg.is_active == UVM_ACTIVE) begin
                mast_sqrh = pll_sequencer::type_id::create("mast_sqrh", this);
                mast_drvh = pll_base_driver::type_id::create("mast_drvh", this);
        end
…
```

PLL Configuration Object

```
typedef enum {PLL1, PLL2, PLL3, PLL4} pll_type_e;

class pll_cfg extends uvm_object;

        pll_type_e pll_type = PLL1;

        //----- Randomization and constraints for PLL frequency of operation -----------
        rand int                feedback_divider;
        rand bit [5:0]          post_divder;
        rand bit [1:0]          ref_divider;

        constraint ref_divider_c {
                ref_divider inside {0,1,2};
        }

        uvm_active_passive_enum is_active = UVM_ACTIVE;
…
```

PLL Base Driver Derived Driver

```
class pll_base_driver extends uvm_driver #(pll_seq_item);
        //phase methods
        …
        //User defined methods
        cold_boot_task(pll_seq_item tr);
        tx_driver();
        …
endclass: pll_base_driver

//Run phase
task pll_base_driver::run_phase(uvm_phase phase);
        tx_driver();
endtask

//User defined methods
task pll_base_driver::tx_driver();
        pll_seq_item  tr;
        seq_item_port.get_next_item(tr);
        case (tr.kind) begin
                pll_seq_item::COLDBOOTPLL : begin
                //Cold Boot method
                end
                pll_seq_item::RESET :  begin
                //Reset method
                …
        endcase
        send(tr);
        seq_item_port.item_done();        …
```

```
class pll_type2_driver extends pll_base_driver;
        extern protected  virtual task cold_boot_task(pll_seq_item tr);
        …
endclass: pll_type2_driver

task pll_type2_driver::cold_boot_task(pll_seq_item tr);
```

PLL Environment

```
class pll_env extends uvm_env;
   pll_tb_cfg tb_cfg;
   pll_agent pll_agnth[*];
```

```
if(tb_cfg.pll_type == PLL1) begin
     for (i=0;i=tb_cfg.no_of_pll_inst; i++) begin
     …
     pll_agent[i]= pll_agent::type_id::create($psprintf("pll_agnth[%0d],i),this);
     end
```

```
if(tb_cfg.pll_type == PLL2) begin
  set_type_override_by_type(pll_base_driver::get_type(), pll_type2_driver::get_type());
  set_type_override_by_type(pll_base_monitor::get_type(), pll_type2_monitor::get_type());
```

```
class pll_env extends uvm_env;
   pll_sb sbh;
   mon_2cov covh;
   pll_tb_cfg tb_cfg;
   pll_agent pll_agnth[*

   extern virtual functi
   ...
endclass: pll_env

function void pll_env::b
   // Get the tb config
   if(!uvm_config_db #(p
     `uvm_fatal(get_type_name(), "Failed to get pll_tb_cfg from the database")
       //Array of configuration objects
   end

   else if (tb_cfg.pll_type == PLL2) begin
     set_type_override_by_type(pll_base_driver::get_type(), pll_type2_driver::get_type());
     set_type_override_by_type(pll_base_monitor::get_type(), pll_type2_monitor::get_type());
       for (j=0; j<tb_cfg.no_of_pll2_inst; j++)
       begin
        uvm_config_db #(pll_cfg)::set(this, $sformatf("pll_agnth[%0d].*",j), "pll_cfg", tb_cfg.pll_cfgh[j]);
        pll_agnth[j] = pll_agent::type_id::create($psprintf("pll_agnth[%0d]",j),this);
       end
```

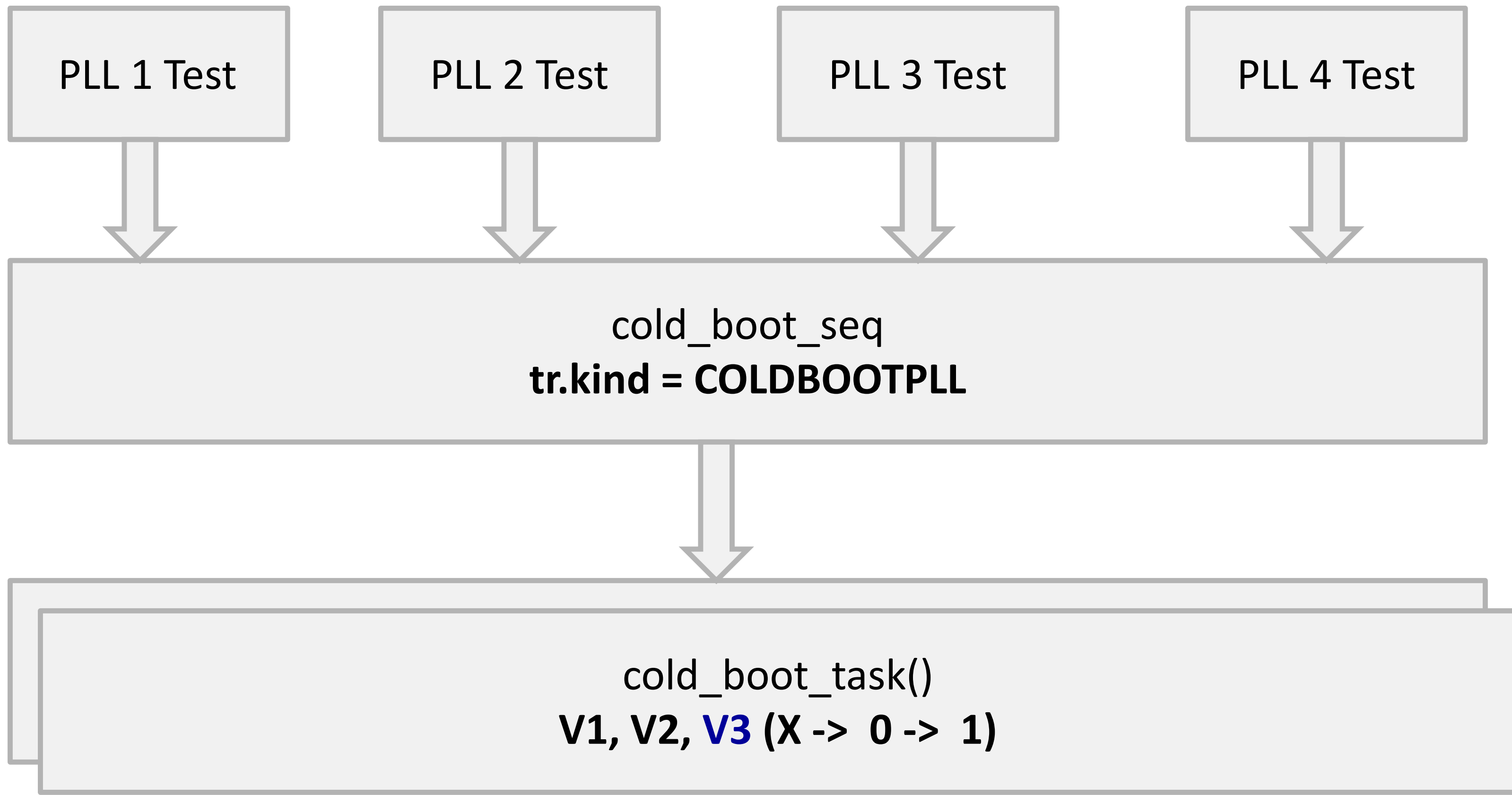
PLL Tests
Common Sequence Library

- PLL Test – base_pll_test
  - Calls sequence libraries for different scenarios
- PLL Sequence library – pll_sequencer_library
  - Generates all scenarios for verifying the PLL variants

```
class base_pll_test extends uvm_test;
  `uvm_component_utils(base_pll_test)

  pll_type_e pll_type;
  int no_of_pll_inst;
  pll_tb_cfg tb_cfg;
  slave_cfg slv_cfgh;
  pll_env envh;

  ...
class pll1_test extends base_pll_test;

  virtual function void build_phase(uvm_phase phase);
    pll_type = PLL1;
    no_of_pll_inst = 1;

    ...
class pll2_test extends base_pll_test;

  virtual function void build_phase(uvm_phase phase);
    pll_type = PLL2;
    no_of_pll_inst = 2;

    ...
```

```
class pll_sequencer_sequence_library extends uvm_sequence_library # (pll_seq_item);


  function new(string name = "simple_seq_lib");
    super.new(name);
    init_sequence_library();
  endfunction

endclass

//Base sequence
class base_sequence extends uvm_sequence #(pll_seq_item);

...
endclass

class cold_boot_pll_seq extends base_sequence;
  `uvm_add_to_seq_lib(cold_boot_pll_seq,pll_sequencer_sequence_library)

   rand kind_e kind1;
   constraint kind1_c {kind1 == COLDBOOTPLL;}
   pll_cfg m_cfg;

   ...
```

Cold Boot PLL Test



AGENDA

- Overview of PLL verification
- Verification Challenges in Prior work
- Proposed Solution
- Outcomes & Conclusion

Outcomes

- ✓ Highly reusable PLL verification suite
  - ✓ Common pool of tests & sequences
  - ✓ Common SVA based checkers
  - ✓ Shared tasks for driving common protocol requirements
- ✓ ~50% reduction in verification bring up time
- ✓ 60% reduction in resource requirements
  - ❑ 1 to 2 resource for any number of variants

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2015

Conclusion

✓Unified Verification Environment Template for the verification of a set of IPs with similar design/functionality

THANK YOU