2024

DESIGN AND VERIFICATION™

DVCON

CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA

MARCH 4-7, 2024

Working within the Parameters that System Verilog has constrained us to

Salman Tanvir, David Crutchfield, Markus Brosch

accellera SYSTEMS INITIATIVE

Motivation

- Parameterized IP is a fundamental building Block in SOCs
- Parameter handling challenging in System Verilog
- DUT to TB connection for Parameterized IP
  - Compare known approaches
  - Propose improvements for emulation compatibility
- UVM Factory
- Coverage

DUT to TB Connection for Parametrized IP

- The parameterized VIF problem
  - HVL VIF handle is type specific
  - Each specialization of a parametrized interface is a new type
- Parameterized class approach
- Maximum Footprint approach
- Polymorphic Interface
- Emulation Compatible Polymorphic Interface

Parameterized Class Approach

- VIF encapsulating class component is parameterized

✓ Multiple class specializations can easily be created in a single environment

✓ Supports parametric functional coverage out of the box

✗ Parameter ripple effect

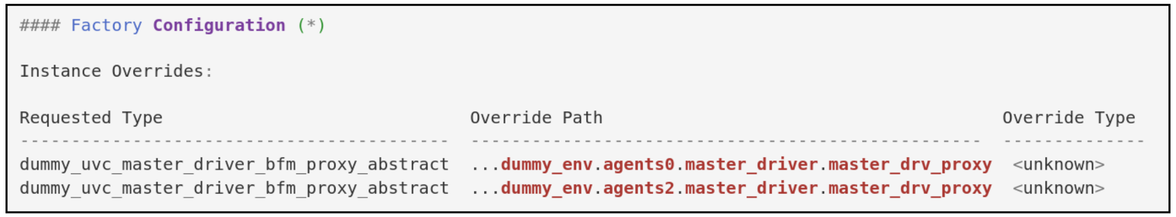
✗ UVM Factory limitations

- Not recommended to solve the parameterized VIF problem
- Powerful tool when used as base classes

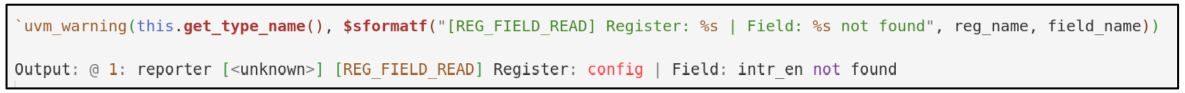
Parameterized Class Approach

UVM Factory Limitations

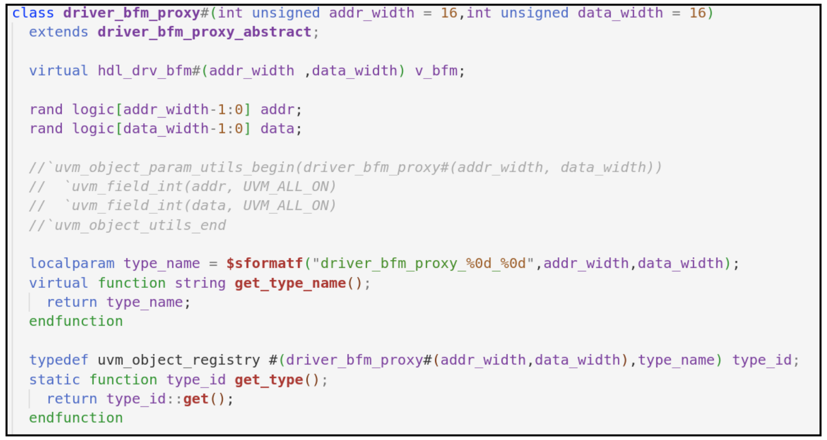
- Utility macros do not support name-based factory
- String type name not declared
- Debug difficult with type-based factory
- Tagging debug messages by type not possible



```
#### Factory Configuration (*)

Instance Overrides:

Requested Type                            Override Path                                          Override Type
----------------------------------------  -----------------------------------------------------  -------------
dummy_uvc_master_driver_bfm_proxy_abstract  ...dummy_env.agents0.master_driver.master_drv_proxy  <unknown>
dummy_uvc_master_driver_bfm_proxy_abstract  ...dummy_env.agents2.master_driver.master_drv_proxy  <unknown>
```



```
`uvm_warning(this.get_type_name(), $sformatf("[REG_FIELD_READ] Register: %s | Field: %s not found", reg_name, field_name))

Output: @ 1: reporter [<unknown>] [REG_FIELD_READ] Register: config | Field: intr_en not found
```



```
class driver_bfm_proxy#(int unsigned addr_width = 16,int unsigned data_width = 16)
  extends driver_bfm_proxy_abstract;

  virtual hdl_drv_bfm#(addr_width ,data_width) v_bfm;

  rand logic[addr_width-1:0] addr;
  rand logic[data_width-1:0] data;

  //`uvm_object_param_utils_begin(driver_bfm_proxy#(addr_width, data_width))
  //  `uvm_field_int(addr, UVM_ALL_ON)
  //  `uvm_field_int(data, UVM_ALL_ON)
  //`uvm_object_utils_end

  localparam type_name = $sformatf("driver_bfm_proxy_%0d_%0d",addr_width,data_width);
  virtual function string get_type_name();
    return type_name;
  endfunction

  typedef uvm_object_registry #(driver_bfm_proxy#(addr_width,data_width),type_name) type_id;
  static function type_id get_type();
    return type_id::get();
  endfunction
```

Maximum Footprint Approach

- Size interfaces to a max width without parameterization
- Use a subset as required in each environment
- Max width defined by macro and can be increased if needed

✓ Avoids the parameter ripple effect and UVM factory issues

✓ Simple and pragmatic

✗ Overhead to specify which part of the interface connects to the DUT

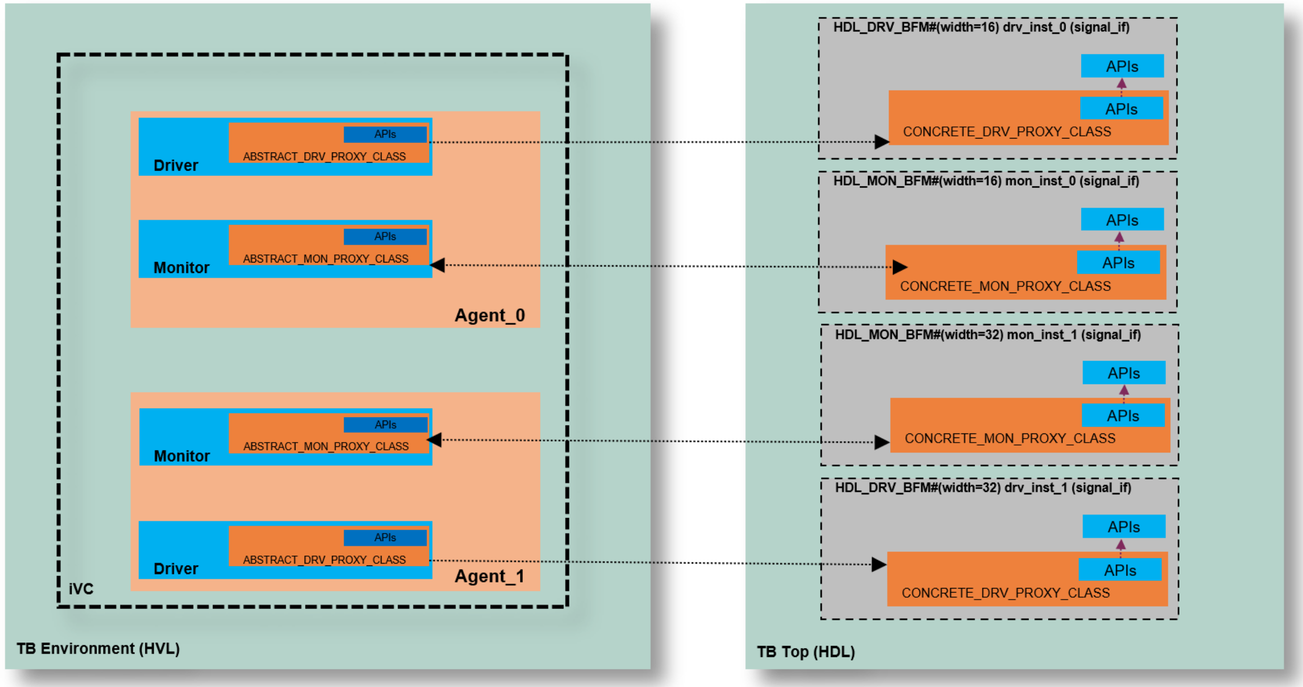
✗ Debug can become difficult if a large part of signal is not utilized

Polymorphic Interface

- Connection via APIs
- Abstract class handles instead of VIF
- Concrete class implements APIs and resides inside the BFMs
- Concrete class is parameter free

✓ Decoupling of class-based side from SV Interface
  - Encapsulation and separation of concerns (OOP)
  - Reusability and extendibility

✗ Not Emulation compatible

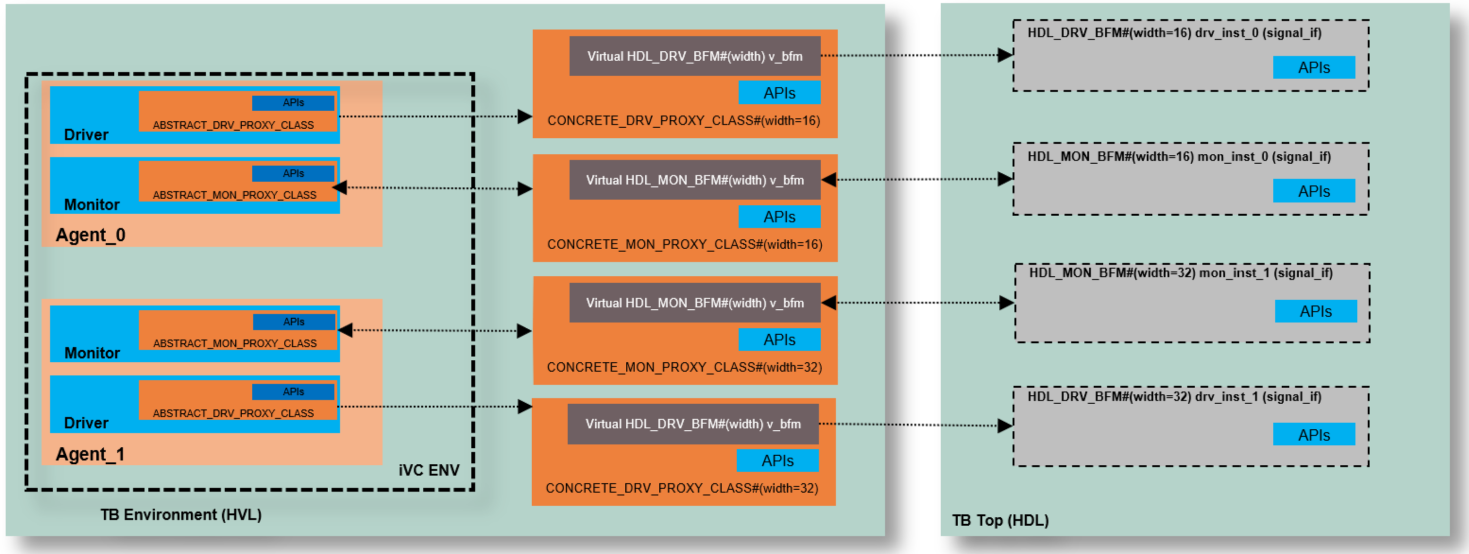


Comparison

Approach	✓	✗
Parameterized class	Solving parametrized VIF and functional coverage problem	Parameter ripple effect UVM Factory limitations
Maximum Footprint	Simple and pragmatic	Overhead for DUT connection Debug can become difficult
Polymorphic Interface	Decoupling of class-based side from SV Interface Reusability and extendibility	Not Emulation compatible

Emulation Ready Polymorphic Interface

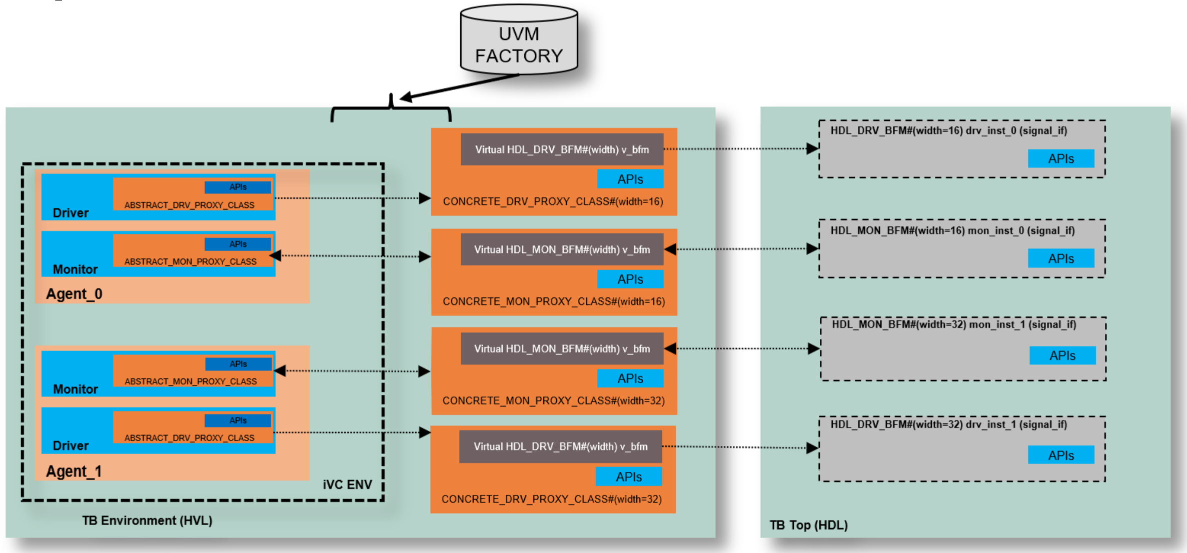
- Emulation requires synthesizable BFMs
- Concrete proxy class relocated to HVL side
  - Contains a VIF handle to the BFMs
  - Needs to be parameterized
  - Abstract handle remains parameter free
  - Concrete proxy class parameters not exposed to encapsulating components



Driver
APIs
ABSTRACT_DRV_PROXY_CLASS
Monitor
APIs
ABSTRACT_MON_PROXY_CLASS
Agent_0
Monitor
APIs
ABSTRACT_MON_PROXY_CLASS
Driver
APIs
ABSTRACT_DRV_PROXY_CLASS
Agent_1
iVC ENV
TB Environment (HVL)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=32)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=32)
HDL_DRV_BFM#(width=16) drv_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=16) mon_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=32) mon_inst_1 (signal_if)
APIs
HDL_DRV_BFM#(width=32) drv_inst_1 (signal_if)
APIs
TB Top (HDL)

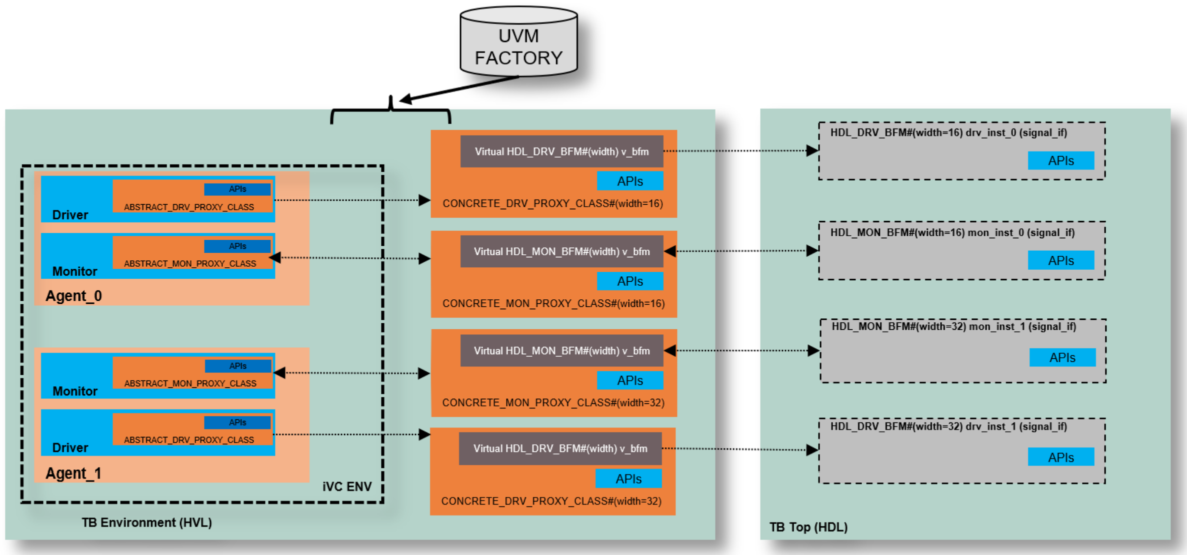
Emulation Ready Polymorphic Interface

- Each specialized proxy must be built and VIFs set
- UVM Factory is an ideal candidate
  - Parameterization challenges remain
  - UVM 1.2 is does not support abstract class registration


UVM FACTORY
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
Agent_0
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
Agent_1
iVC ENV
TB Environment (HVL)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=32)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=32)
HDL_DRV_BFM#(width=16) drv_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=16) mon_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=32) mon_inst_1 (signal_if)
APIs
HDL_DRV_BFM#(width=32) drv_inst_1 (signal_if)
APIs
TB Top (HDL)

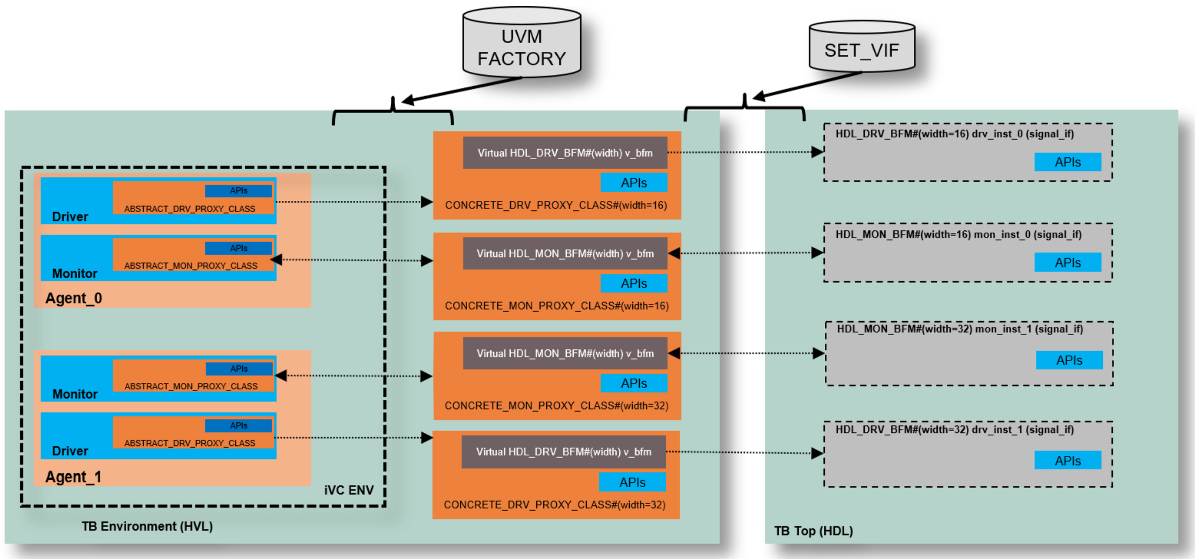
Emulation Ready Polymorphic Interface

```
class uvc_env extends uvm_env;
  uvc_agent agents[];
endclass: uvc_env

class tb_env extends uvm_env;
  uvc_env uvc_env;
endclass: tb_env

typedef driver_bfm_proxy# (16, 16) driver_bfm_proxy16;
typedef driver_bfm_proxy# (32, 32) driver_bfm_proxy32;

class tb_base_test extends uvm_test;
  tb_env tb_env;

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

// Factory overrides
set_inst_override_by_type("tb_env.uvc_env.agents0.driver.drv_proxy",
  driver_bfm_proxy_abstract::get_type(), driver_bfm_proxy16::get_type() );
set_inst_override_by_type("tb_env.uvc_env.agents2.driver.drv_proxy",
  driver_bfm_proxy_abstract::get_type(), driver_bfm_proxy32::get_type() );

  ...
```


UVM FACTORY
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
Agent_0
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
Agent_1
iVC ENV
TB Environment (HVL)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=32)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=32)
HDL_DRV_BFM#(width=16) drv_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=16) mon_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=32) mon_inst_1 (signal_if)
APIs
HDL_DRV_BFM#(width=32) drv_inst_1 (signal_if)
APIs
TB Top (HDL)

Emulation Ready Polymorphic Interface

- Factory only creates concrete proxies
- VIF setting required
  - Can be generically automated


UVM FACTORY
SET_VIF
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=32)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=32)
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
Agent_0
APIs
ABSTRACT_MON_PROXY_CLASS
Monitor
APIs
ABSTRACT_DRV_PROXY_CLASS
Driver
Agent_1
iVC ENV
TB Environment (HVL)
HDL_DRV_BFM#(width=16) drv_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=16) mon_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=32) mon_inst_1 (signal_if)
APIs
HDL_DRV_BFM#(width=32) drv_inst_1 (signal_if)
APIs
TB Top (HDL)

```
class set_vif_util#(type V_BFM_T=int , type PROXY_T=int, type ABSTRACT_PROXY_T=int);
  static PROXY_T proxy_handle;
  static V_BFM_T vif_handle;
  static function bit set_vif( ABSTRACT_PROXY_T abstract_proxy,string scope = "", name ="");

    if (!$cast(proxy_handle,abstract_proxy))
      `uvm_fatal("FATAL CAST", "Incompatible types for cast!");

    if(!uvm_resource_db #(V_BFM_T)::read_by_name(scope, name , vif_handle))
      `uvm_fatal("FATAL V_BFM", "Virtual BFM config_db retrieval failed!");

    proxy_handle.v_bfm=vif_handle;

  endfunction

endclass
```

```
module hdl_tb_top;

  initial begin
    uvm_resource_db #(virtual hdl_drv_bfm#(16,16))::set("*", "drv0",drv0);
    uvm_resource_db #(virtual hdl_drv_bfm#(32,32))::set("*", "drv2",drv2);
  end
endmodule

function void tb_base_test::connect_phase(uvm_phase phase);
  super.connect_phase(phase);

    set_vif_util#(virtual hdl_master_drv_bfm# (16,16),driver_bfm_proxy16,
      driver_bfm_proxy_abstract)::set_vif(uvc_env.agent[0].drv_proxy,"*","drv0");
    set_vif_util#(virtual hdl_master_drv_bfm# (32,32),driver_bfm_proxy32,
      driver_bfm_proxy_abstract)::set_vif(uvc_env.agent[2].drv_proxy,"*","drv2");

...
endfunction
```

Emulation Ready Polymorphic Interface

- UVM 1.2 Workarounds
  - Make abstract class non-virtual
  - APIs can no longer be pure and must be implemented
  - Base class APIs must not be called directly

```
class driver_bfm_proxy_abstract extends uvm_object;

  `uvm_object_utils(driver_bfm_proxy_abstract)

  virtual function void error_handler(string tag = "error_handler" );
      `uvm_fatal({"PURE_BASE_CLASS: ",tag}, "This API should not be called directly! Please fix factory override");
  endfunction

  virtual task automatic do_drive(drv_item tr_arg);
    error_handler("do_drive");
  endtask
```

Emulation Ready Polymorphic Interface

- Alternative factory free approach

✓ Avoids the UVM factory issue

✓ Simpler for novice users

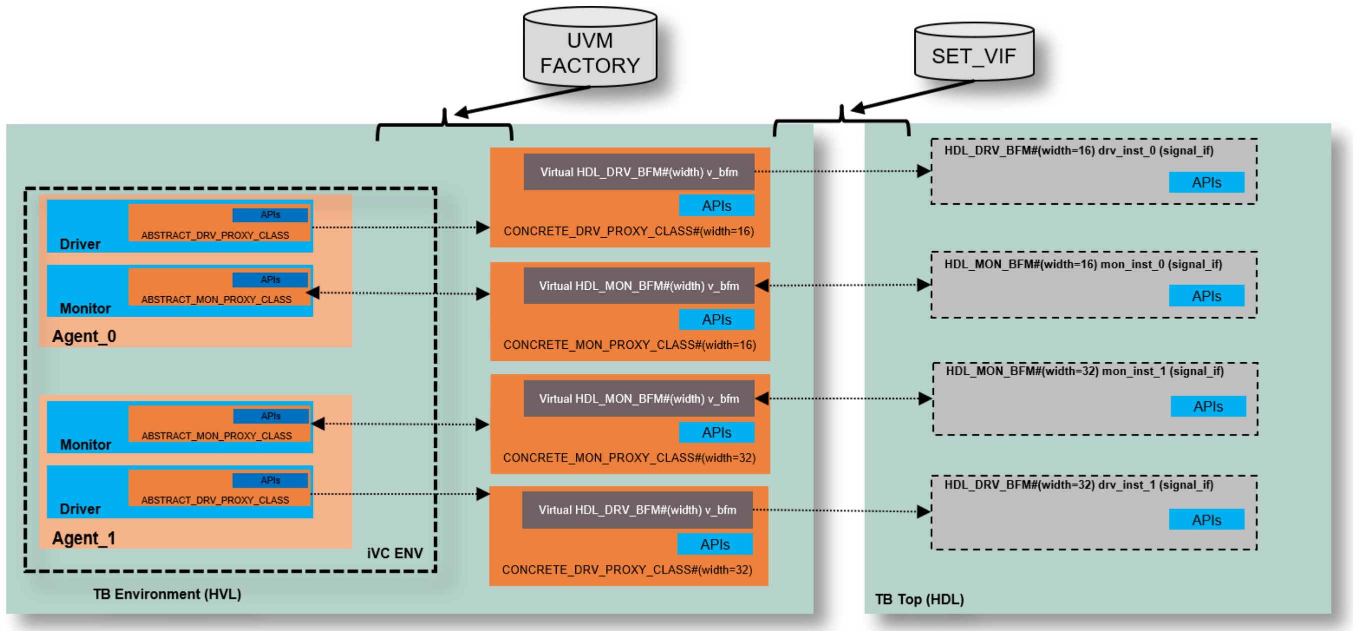
✓ Easy VIF setting

✗ Lose factory automation

```
module hdl_tb_top;

  signals_if#(16,16)         s_if();
  hdl_master_drv_bfm#(16,16) master_drv(s_if);
...

endmodule

module hvl_tb_top;
  master_driver_bfm_proxy#(16,16) concrete_proxy;

  initial begin
    concrete_proxy = new();
    concrete_proxy.v_bfm = hdl_tb_top.master_drv;
    uvm_resource_db #(master_driver_bfm_proxy_abstract)::set("*", "master_drv",concrete_proxy);
    run_test();
  end
endmodule

function void tb_base_test::connect_phase(uvm_phase phase);
  super.connect_phase(phase);

  if(!uvm_resource_db #(master_driver_bfm_proxy_abstract)::read_by_name("",
     "master_drv", tb_env.uvc_env.agents[0].master_driver.master_drv_proxy))
    `uvm_fatal("FATAL V_BFM", "Virtual BFM config_db retrieval failed!");

...
endfunction
```

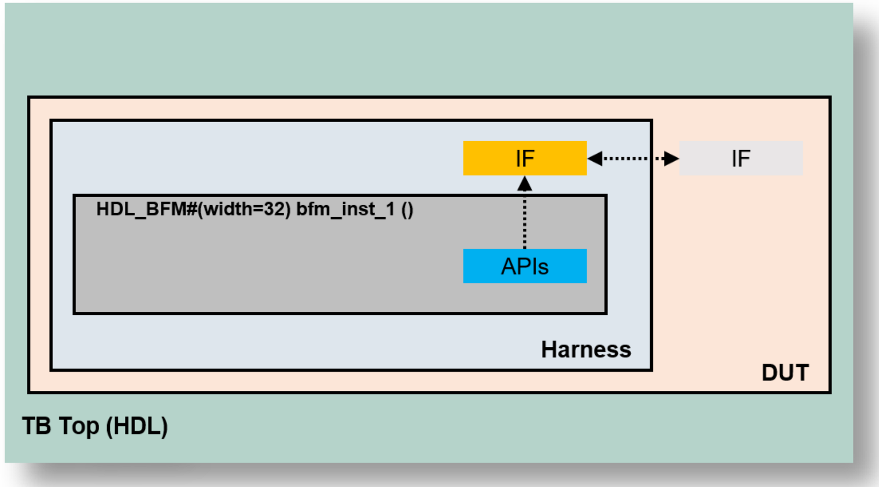
Parameterized Virtual Interface

- Parameterized DUT interface passed to BFMs as a port argument
  - Works with most EDA tools but with a warning
  - LRM: *“Although an interface may contain hierarchical references to objects outside its body or ports that reference other interfaces, it shall be illegal to use an interface containing those references in the declaration of a virtual interface”*
  - This rule should be revised to specify concrete restrictions
  - Harness approach proposed as a solution to be future proof


UVM FACTORY
SET_VIF
Driver
APIs
ABSTRACT_DRV_PROXY_CLASS
Monitor
APIs
ABSTRACT_MON_PROXY_CLASS
Agent_0
Monitor
APIs
ABSTRACT_MON_PROXY_CLASS
Driver
APIs
ABSTRACT_DRV_PROXY_CLASS
Agent_1
iVC ENV
TB Environment (HVL)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=16)
Virtual HDL_MON_BFM#(width) v_bfm
APIs
CONCRETE_MON_PROXY_CLASS#(width=32)
Virtual HDL_DRV_BFM#(width) v_bfm
APIs
CONCRETE_DRV_PROXY_CLASS#(width=32)
HDL_DRV_BFM#(width=16) drv_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=16) mon_inst_0 (signal_if)
APIs
HDL_MON_BFM#(width=32) mon_inst_1 (signal_if)
APIs
HDL_DRV_BFM#(width=32) drv_inst_1 (signal_if)
APIs
TB Top (HDL)

Parameterized Virtual Interface

- Harness approach
  - Encapsulate BFMs and interfaces in a harness
  - Use upwards references to access interface
  - Bind harness into DUT



```
interface harness#(int addr_width = 16, data_width = 16);

  signals_if dut_if(.data(dut.data), .addr(dut.addr)):
  hdl_mon_bfm#(addr_width , data_width) mon_bfm_inst();
  hdl_drv_bfm#(addr_width , data_width) drv_bfm_inst();

endinterface

bind dut harnesss(.addr_width(addr_width), .data_width(data_width)) harness_inst();
```

Coverage for Parameterized IP

- Parameterized classes
    - Out of the box coverage
    - Ripple effect
- Dynamic fields instead of static design parameters
    - Pass to constructor of generic cover groups
    - Must deal with configuration phasing problem
    - Cover groups must be created in the encapsulating class's constructor
    - A possible solution is to encapsulate coverage in an arbitrary class
- Polymorphic coverage using the UVM factory
    - Use UVM Factory to override a non-parameterized base coverage class

Conclusion

- Parameter handling comes with significant challenges for verification
- Special care required with UVM Factory
- Non-polymorphic VIF complicates parameterized interface handling
- The polymorphic interface approach stands out
  - Encapsulation and separation of concerns (OOP)
  - Reusability and extendibility
  - Enhanced to support Emulation
- Parameters have implications on coverage
  - Polymorphic coverage approach recommended

Questions?