2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Highly Configurable UVM Environment for Parameterized IP Verification

HongLiang Liu andy.liu@amd.com

Karl Whiting karl.whiting@amd.com

AMD

Abstract

- This paper describes parameterized AMBA Fabric IP verification.
  - AMBA Fabric connects a group of AMBA master and slave devices.
  - numbers of masters, slaves
  - bus protocols
  - address and data widths

Problem Statement

IP Design — **Always Sync** — **IP Verification**

Many Design Parameters		Test Bench Need Parameters Information
Project Reuse		Project Reuse
Easy Change		Save Verification Effort

IP Change Parameters → **DV Busy With Verification Environment Modification** ?

Prior Work

In order to support one IP's verification in multiple projects , usually two methods:

1) Many Netlists

```
testbench_copy1 PROJECT_1--parameter group1

testbench_copy2 PROJECT_2--parameter group2
```

maintain effort is significant

2) Text Macros

```
FCH Macro
 `ifdef project_A
  `define svt_axi_addr_width 32
 `ifdef project_B
  `define svt_axi_addr_width 64

GPU Macro
 `ifdef project_A
  `define svt_axi_addr_width 64
 `ifdef project_B
  `define svt_axi_addr_width 128
```

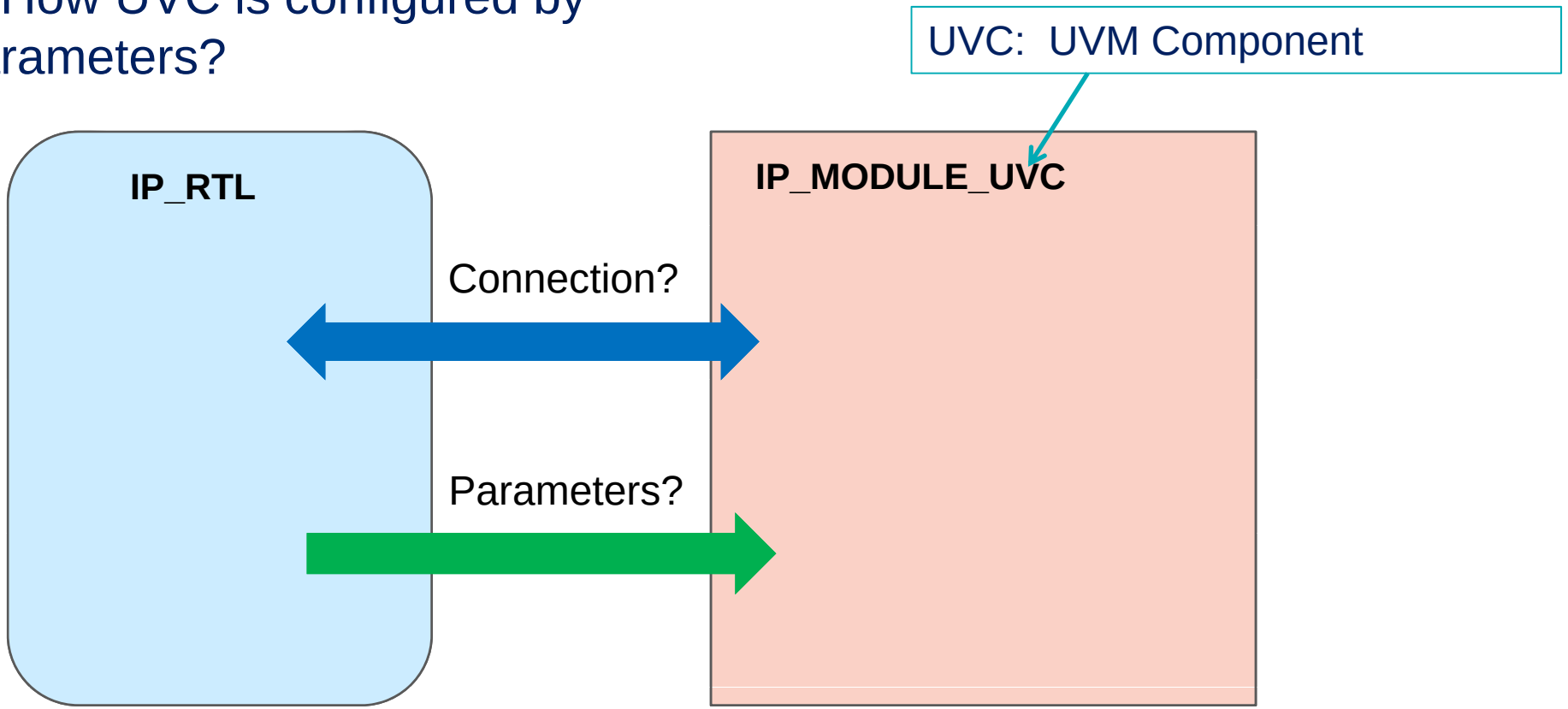
reuse is poor

scope control is a problem

UVM is enough? NO!

1.How connection is controlled by parameters?

2.How UVC is configured by parameters?



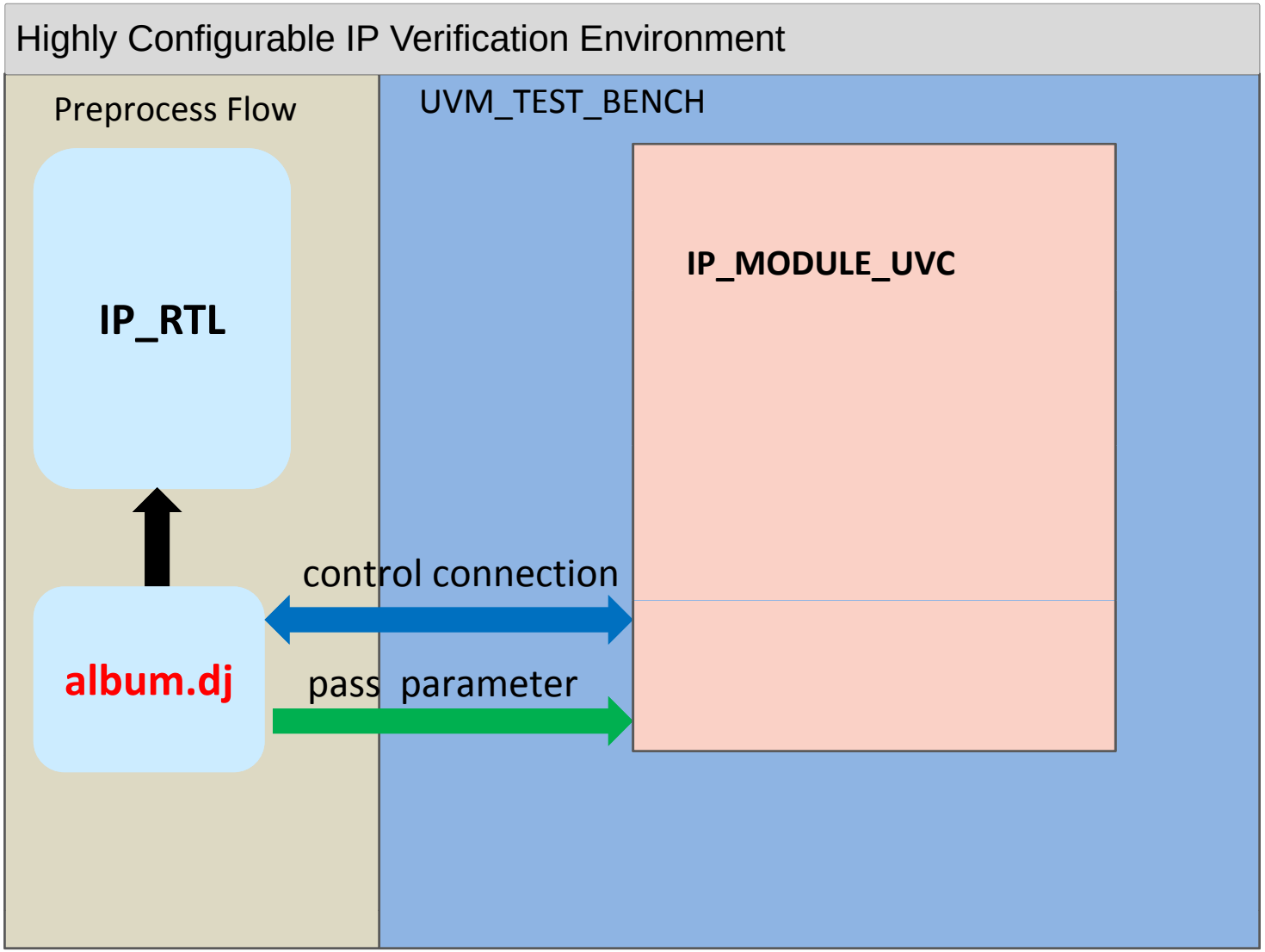
Proposed Approach / Solution

Steps to create highly configurable IP verification environment.

- Step 1: Create Features.
  1) Features/album.dj.
  2) preprocess mechanism.
- Step 2: Connection Bind and Feature Class.
  1) Connection bind.
  2) Feature ruby program.
  3) Feature class structure.
- Step 3: Develop UVM Environment
  1) Highly configurable UVM environment structure
  2) Stimulus sequence and sequencer highly configurable.
  3) Cover group configuration by feature.
  4) Vertical reuse
  5) Horizontal reuse .

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment



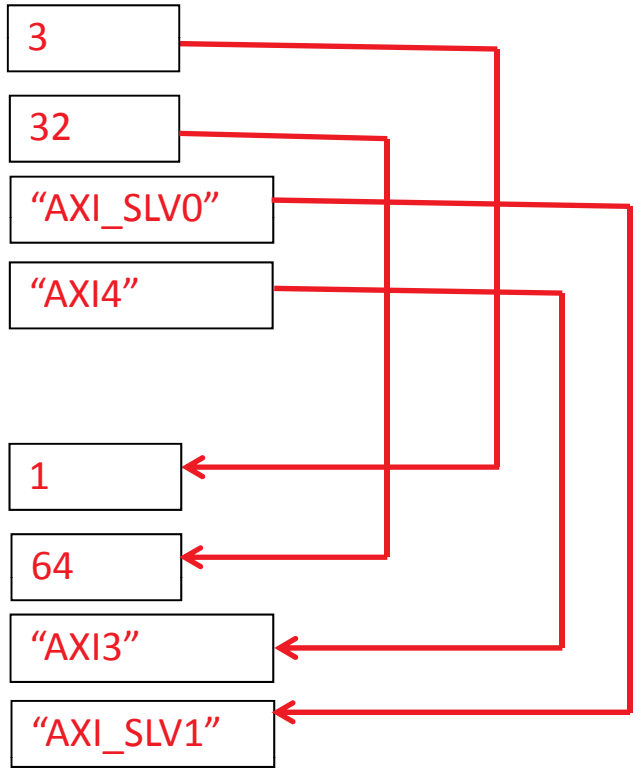
Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

1) Features/album.dj

a)Features are **IP design parameters** , design engineer defines them.

```
component :amba do |variant|
case variant
when: IP_existing
        feature "axi_slaves",
        feature "axi_slave0_addr_width",
        feature "axi_slave0_name",
        feature "axi_slave0_protocol",
        …
 when: IP_future
        feature "axi_slaves",
        feature "axi_slave0_addr_width",
        feature "axi_slave0_protocol ",
        feature "axi_slave0_name ",
```



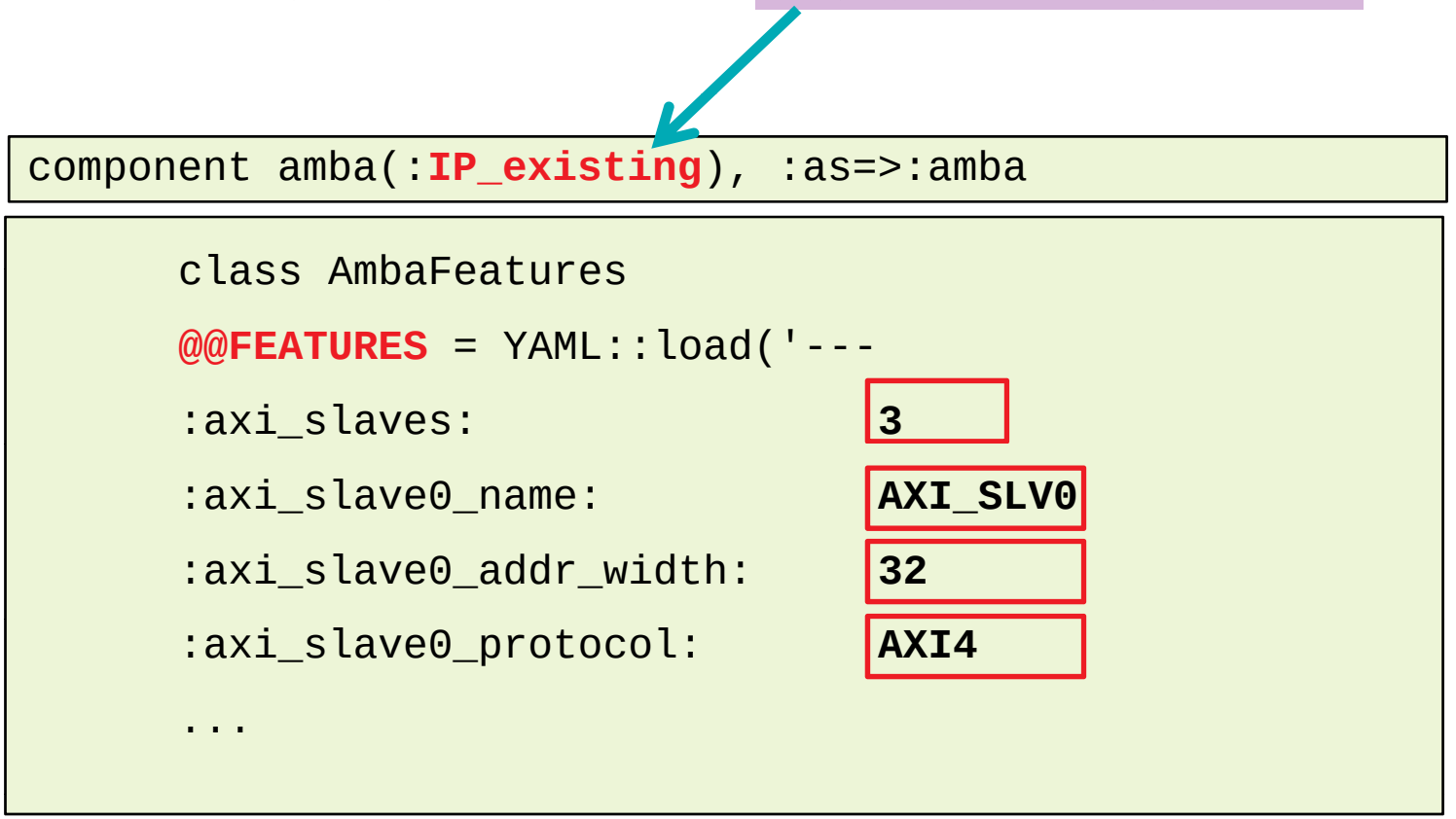
Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

2) preprocess mechanism

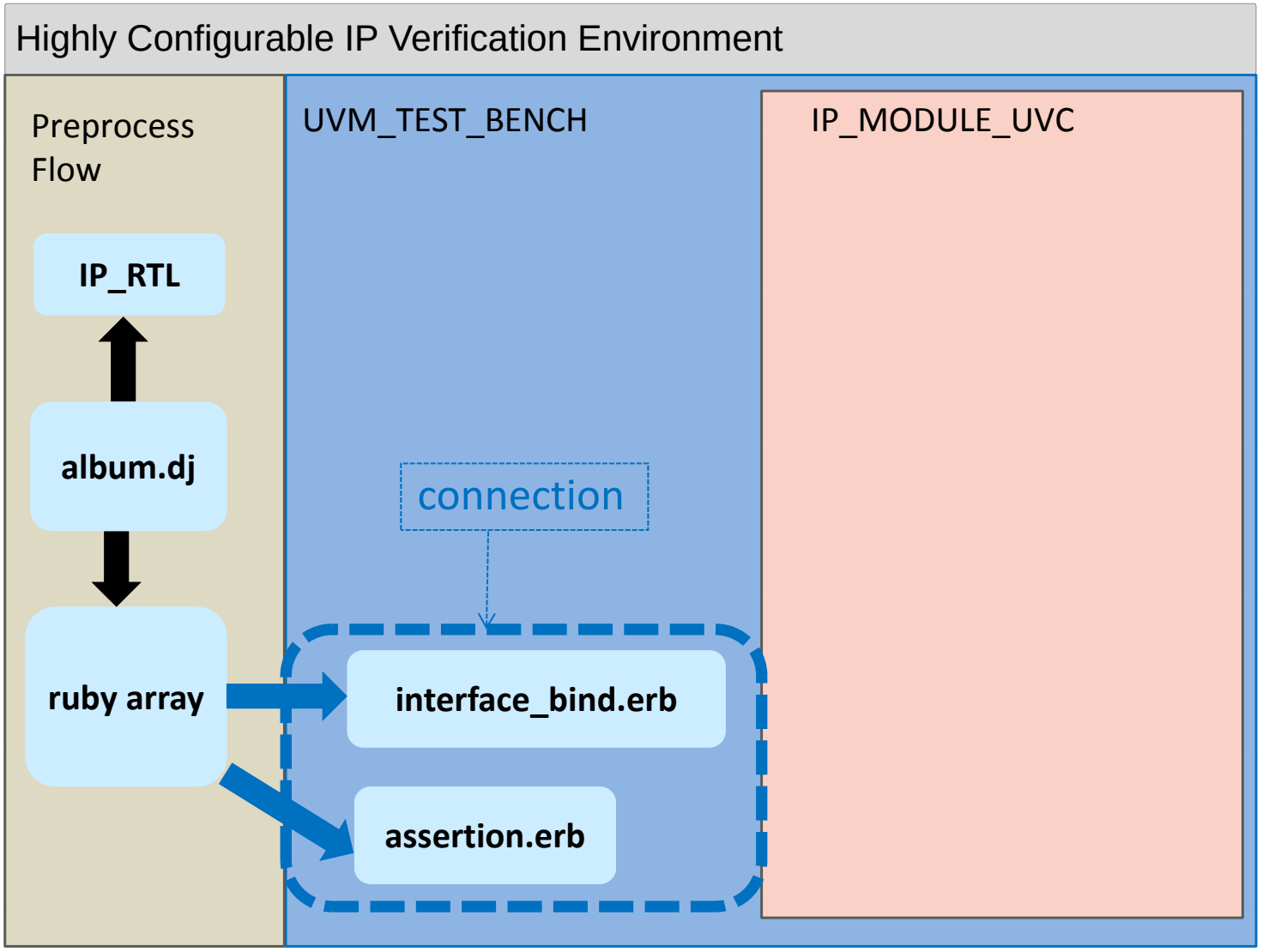
a) preprocess flow translates IP features to ruby format.

b) Generate ruby array **@@FEATURES** of one project provided by argument .



Proposed Approach / Solution

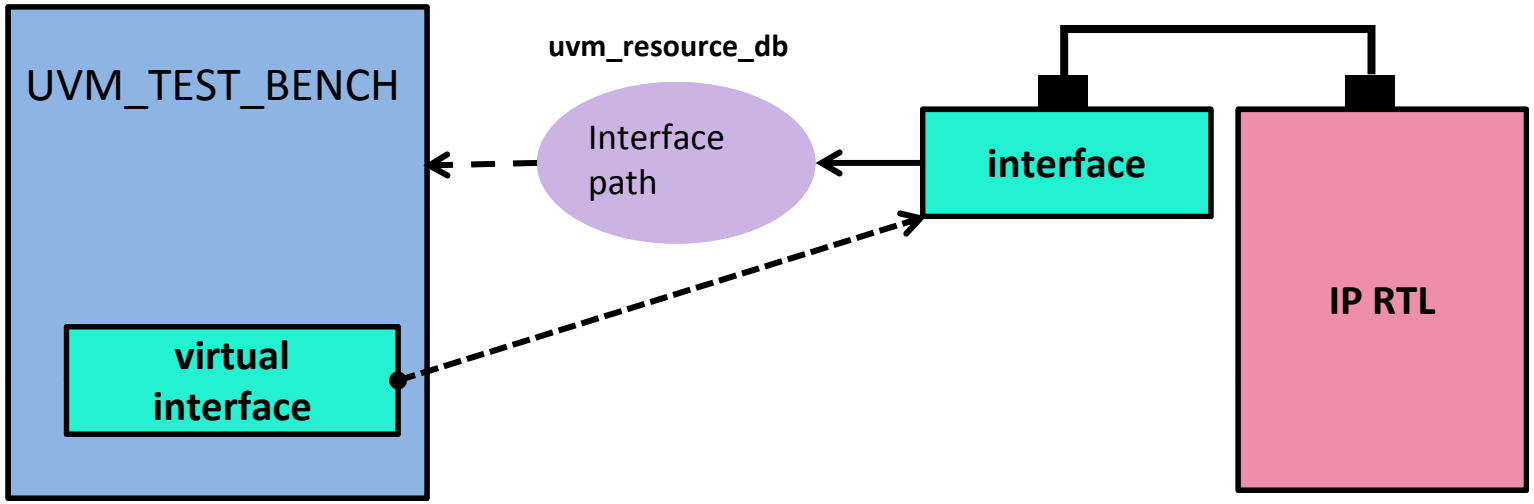
Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment



Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

1) Connection bind
   a) Connect **static RTL world** and **dynamic UVM world** .



Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

b) Interface Bind Ruby Program

```
% for i in 0..(features("axi_slaves")-1) do
bind `IP_AMBA_MODNAME svt_axi_slave_if amba_axi_slave_if<%=i%> (
  .araddr( XMST_<%=features("axi_slave#{i}_port")%>_rdAddr_araddr}),
   ...
%end
```

```
bind `IP_AMBA_MODNAME svt_axi_slave_if amba_top_axi_slave_if0 (
 .araddr(XMST_AXI_SLV0_rdAddr_araddr),
 ...
```

```
bind `IP_AMBA_MODNAME svt_axi_slave_if amba_top_axi_slave_if2 (
 .araddr(XMST_AXI_SLV0_rdAddr_araddr),
 ...
```

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

c) Assertion Bind Ruby Program.

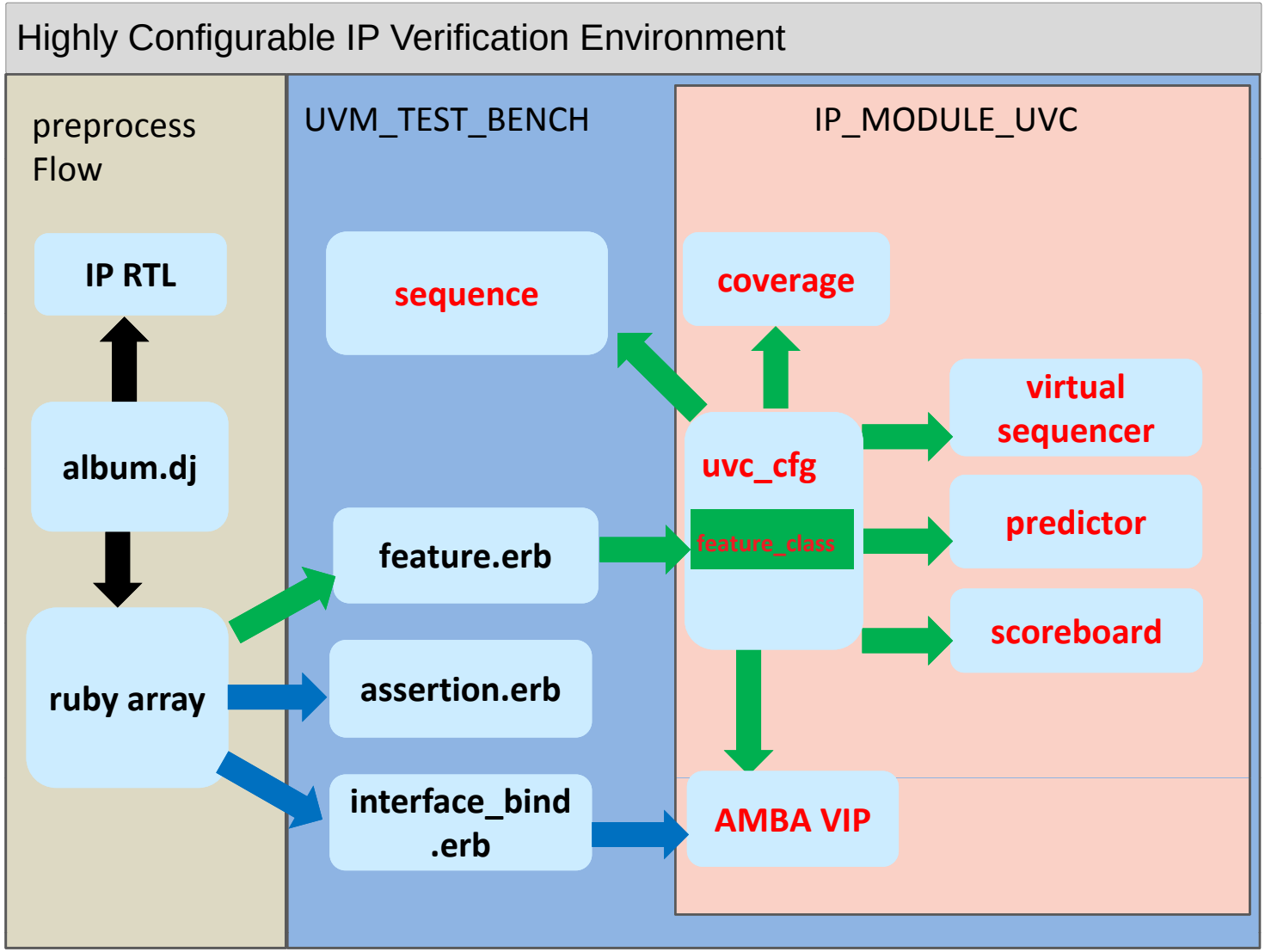
```
bind `IP_AMBA_MODNAME amba_checker amba_checker(
.apb_timeout_clk_in (32'd<%=features("apb_slave_timeout")%>)
...
```



```
 interface    amba_checker ();
property signal_apb0_timeout_cnt_cov;
$rose(start) |-> (($past(apb0_cnt, 1) >= (apb_timeout_clk_in -1))
endproperty
```

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment



Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

2) Feature ruby program

```
package amba_features_pkg;
  class amba_features extends uvm_object;
```

import wherever need

```
        int axi_slv_addr[];
```

① user defines data structure

```
        %AmbaFeatures::features.each_pair do |k,v|
           %if(v.class == String || v.class == Symbol)
           string <%=k%>;
           %else
           int <%=k%>;
        %end
```

② ruby program generates parameter list

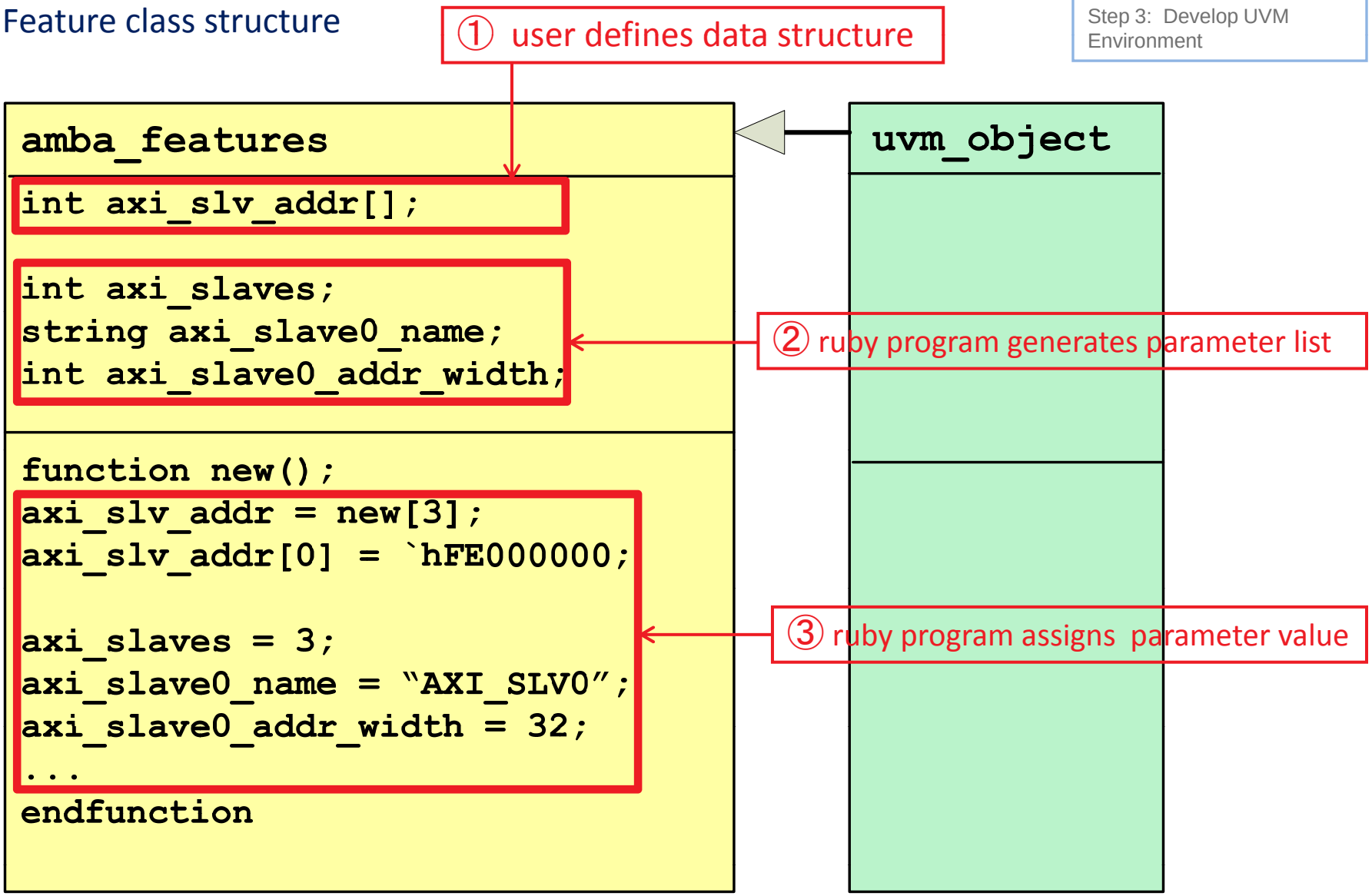
```
        %AmbaFeatures::features.each_pair do |k,v|
        %if "#{k}" =~ /axi_slave\d+.*address/
              axi_slv_addr[<%=i%>] = 'h<%=v%>;
        % i=i+1;
        %end
```

③ ruby program assigns parameter value

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

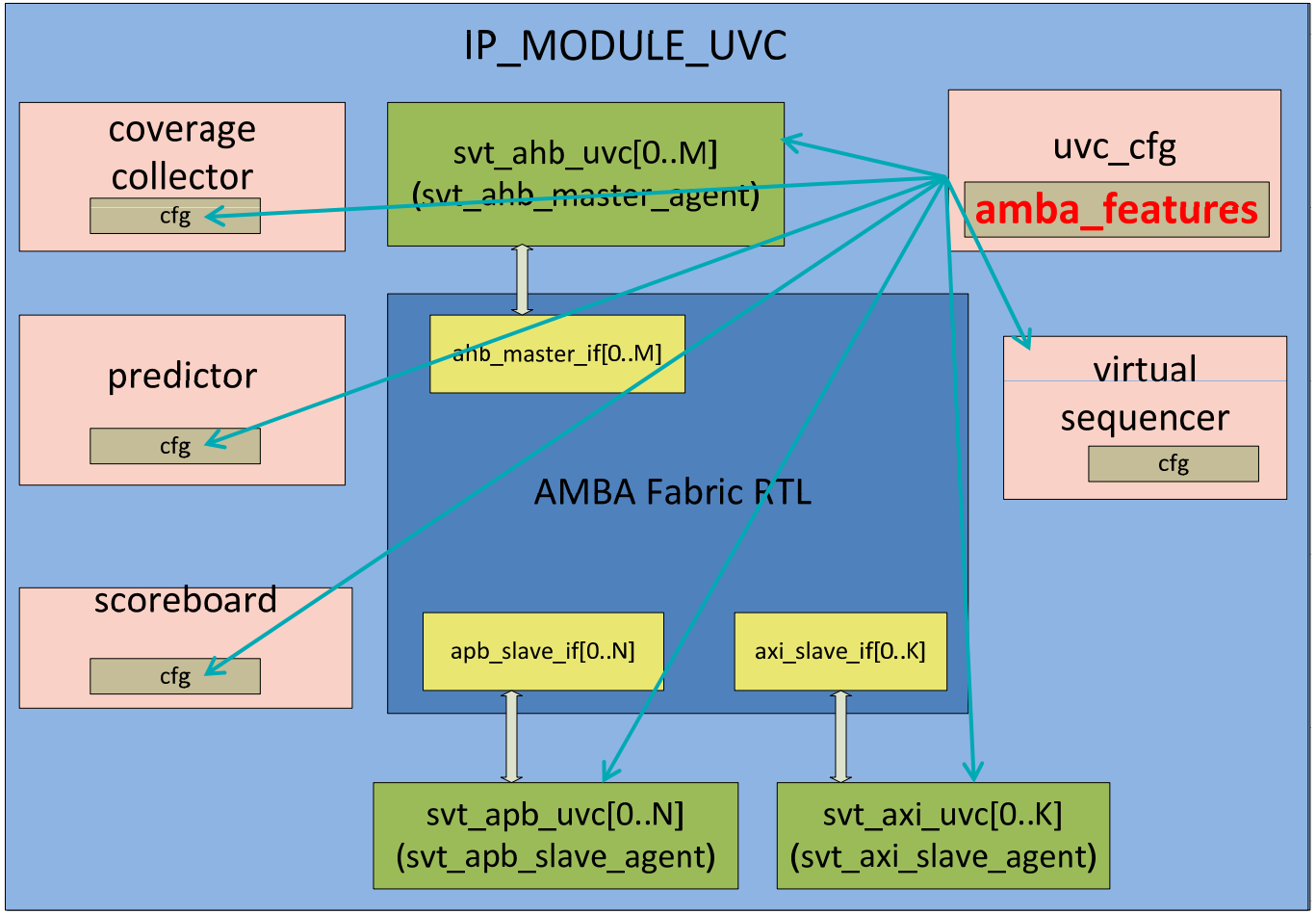
3) Feature class structure



Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

1) Self-adaption UVM environment structure



Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

a) uvc_cfg configure.

```
axi_env_cfg = new[amba_features.axi_slaves];
for (int i=0; i<amba_features.axi_slaves; i++) begin
axi_env_cfg[i].addr_width = amba_features.axi_addr_width[i];
. . .
end
```

b) third-party vip creation configure.

```
svt_axi_vip   axi_uvc_env[];
for (int i=0; i < cfg.amba_top_features.axi_slaves; i++) begin
axi_uvc_env[i]=svt_axi_env::type_id::create(axi_uvc_env[i], this);
end
```

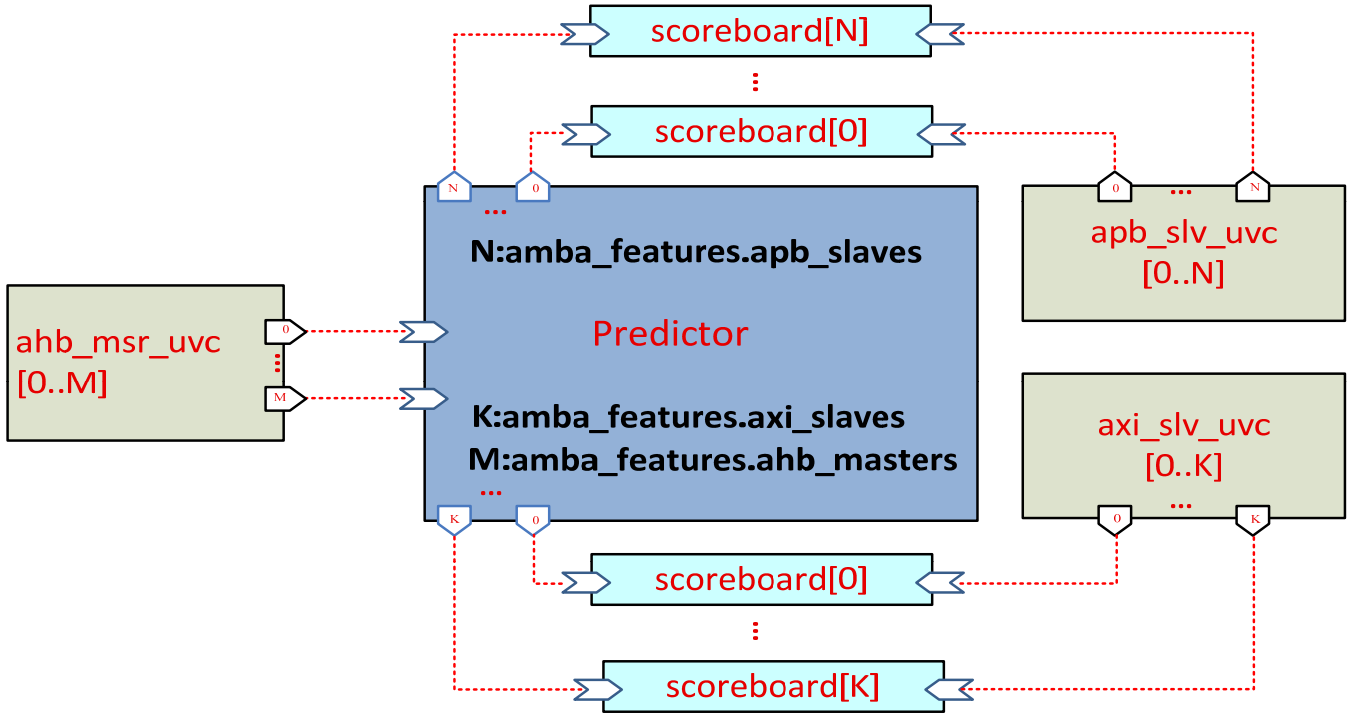
Proposed Approach / Solution

STEP3: DEVELOP UVM environment(1)

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

c) predictor and scoreboard configure.

➢ predictor and scoreboard overview.




```
apb_scoreboard=new[cfg.amba_features.apb_slaves];
axi_scoreboard=new[cfg.amba_features.axi_slaves];
```

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

2) stimulus sequence and sequencer configure.

- virtual sequence start configure.

```
for(int k=0;k<cfg.amba_features.axi_slaves;k++) begin
fork
`uvm_do_on ( amba_axi_seq[k],p_sequencer.axi_slv_sqr[k])
join_none
end
```

- sequence constraint configure

```
for(int k=0;k<cfg.amba_features.axi_slaves;k++) begin
`uvm_rand_send_with (read_tran,
{ read_tran.addr ==  cfg.amba features.axi slv addr[k]+ 'hFFC;
} )
end
```

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

3) cover group configuration by feature.

```
amba_slv_cov_cb
```

```
wr_max = amba_features.axi_slv_outstanding_wr;

coverpoint  num_wr_outstanding
  { bins nums[] = {[0:wr_max]};
```

Be Careful

```
amba_slv_cov_cb axi_slv_callback[];
```

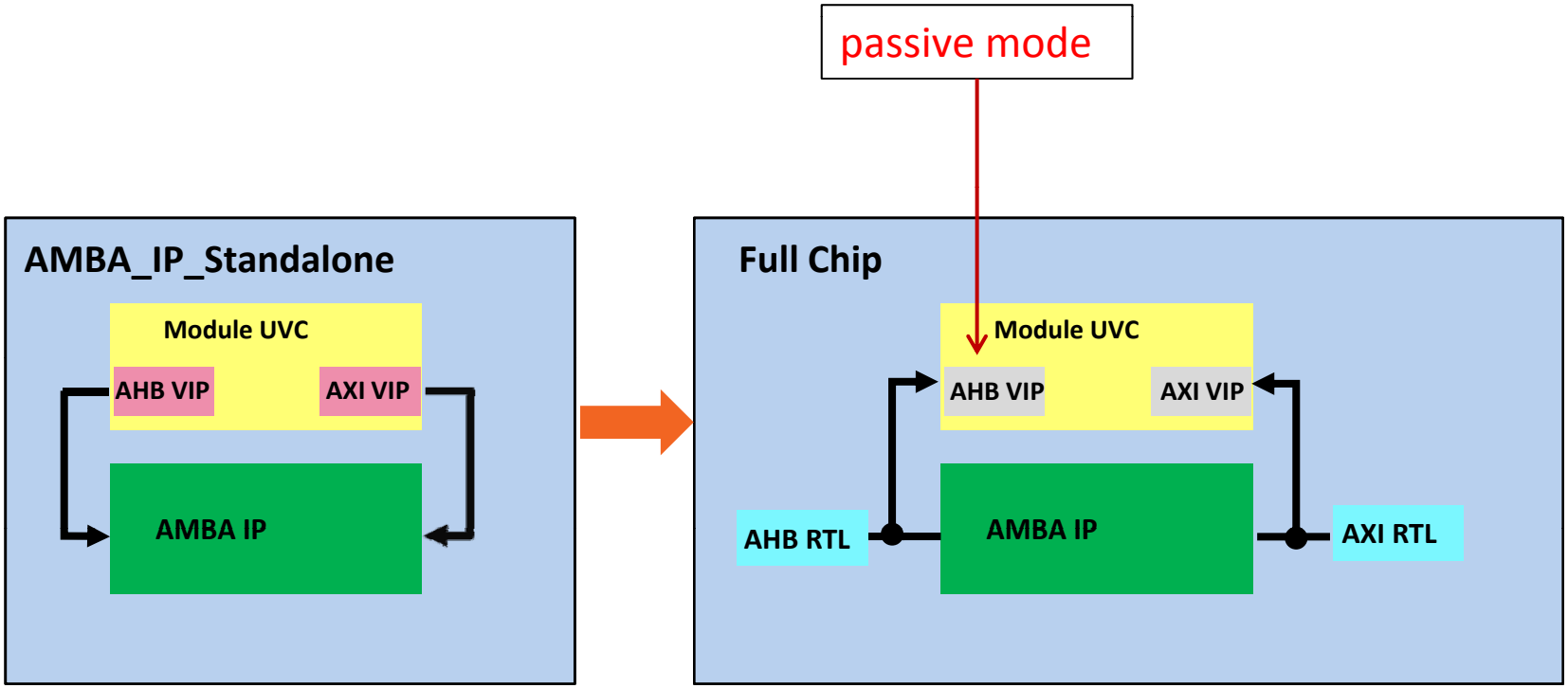
```
axi_slv_callback = new[amba_features.axi_slaves];

foreach(axi_slv_callback[i])  begin
  add_callback(axi_slv_callback[i]);
end
```

Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

4) vertical reuse

From IP level to SOC level

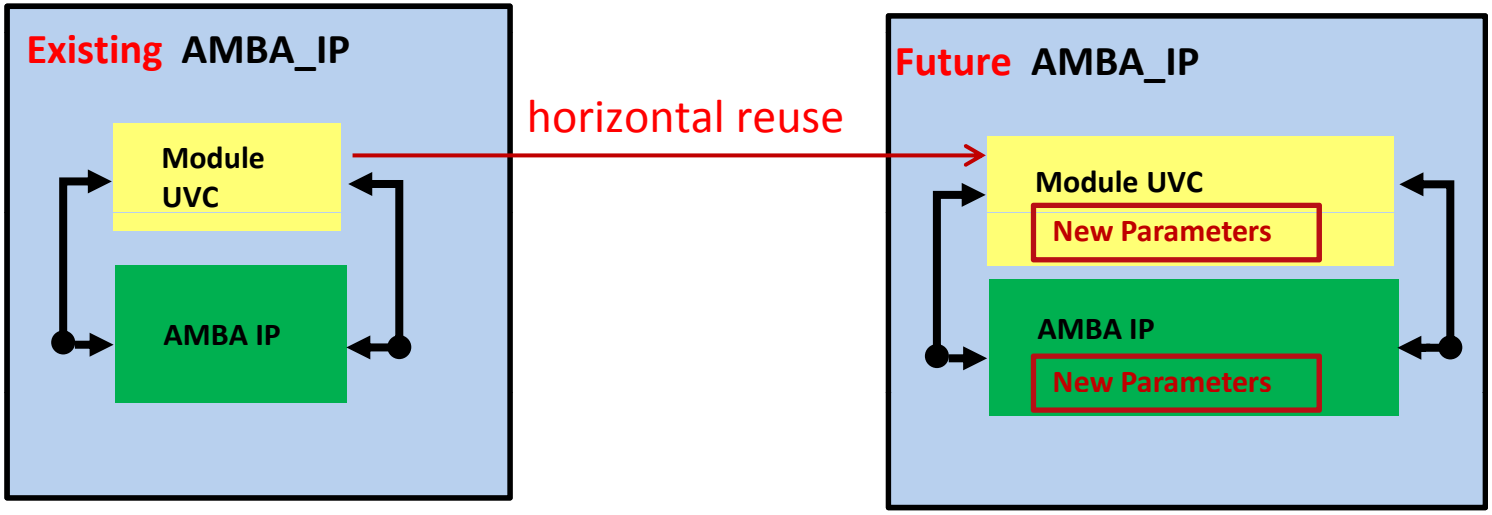


Proposed Approach / Solution

Step 1: Create Features.
Step 2: Connection Bind and Feature Class
Step 3: Develop UVM Environment

5) horizontal reuse

From Project to Project



Results And Conclusions

AMBA IP Change Parameters → **Verification Environment Update Automatically !**

Test bench Connection	**Add or remove connection with parameters**
UVM Component Hierarchy	**Control UVM component build with parameters**
Stimulus	**Issue proper stimulus with parameters**
Predictor	**Predict function with parameters**
Scoreboard	**Compatible with AXI/AHB/APB**
Functional Coverage	**Add or remove coverage with parameters**
Assertion	**Configure timing with parameters**