

# Scoreboard design using method ports

Craig Deaton

Texas instruments

Pandy Kalimuthu

WIPRO Technologies

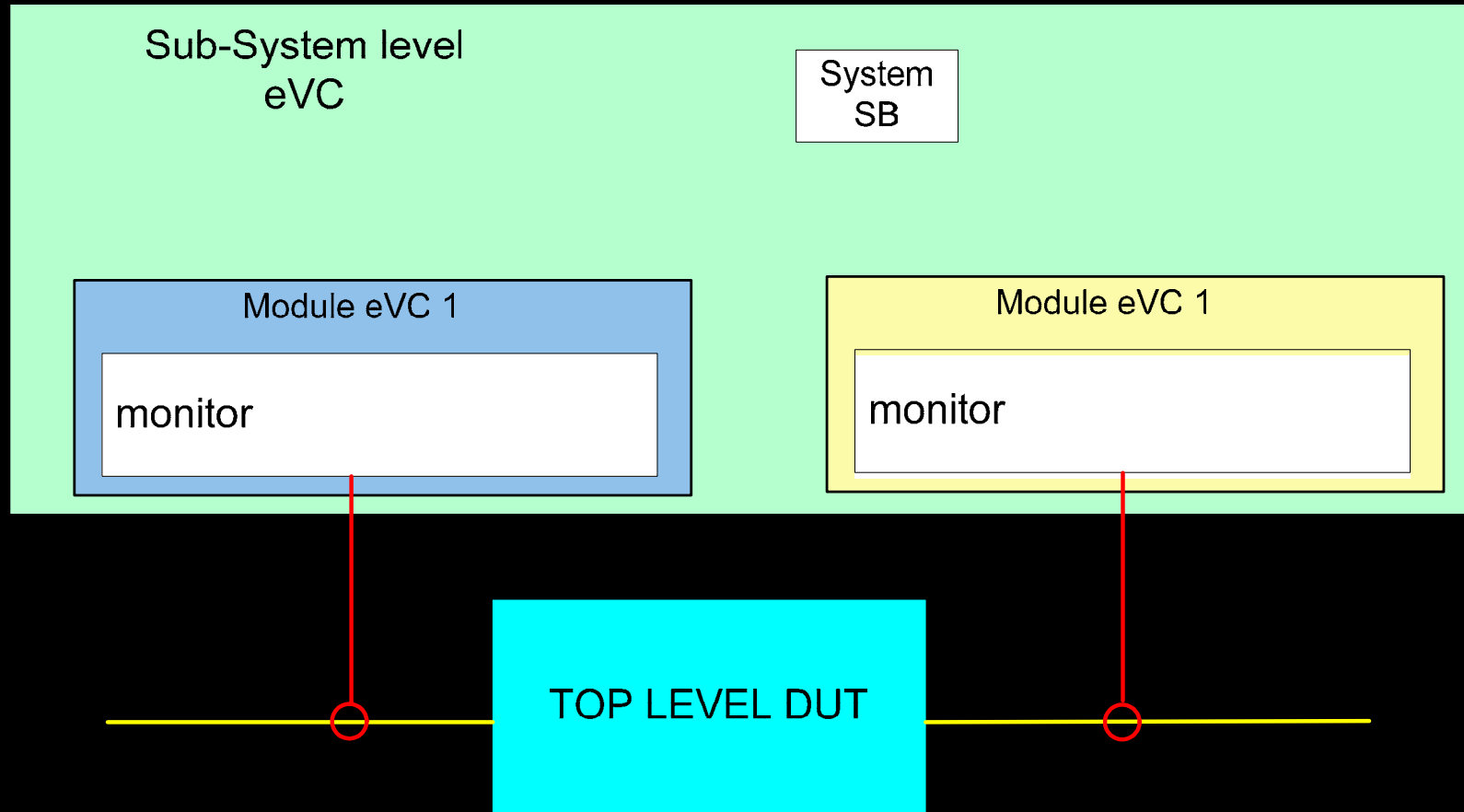
# Introduction

- Method ports provide a mechanism for calling methods or TCMs defined in other 'e' units or written in other foreign languages
- Method ports provide a mechanism for implementing transaction-level interfaces between specman and higher level language
- Here, we have used the method port to construct a scoreboard and demonstrated how powerful they are in terms of exchanging information across various eVCs

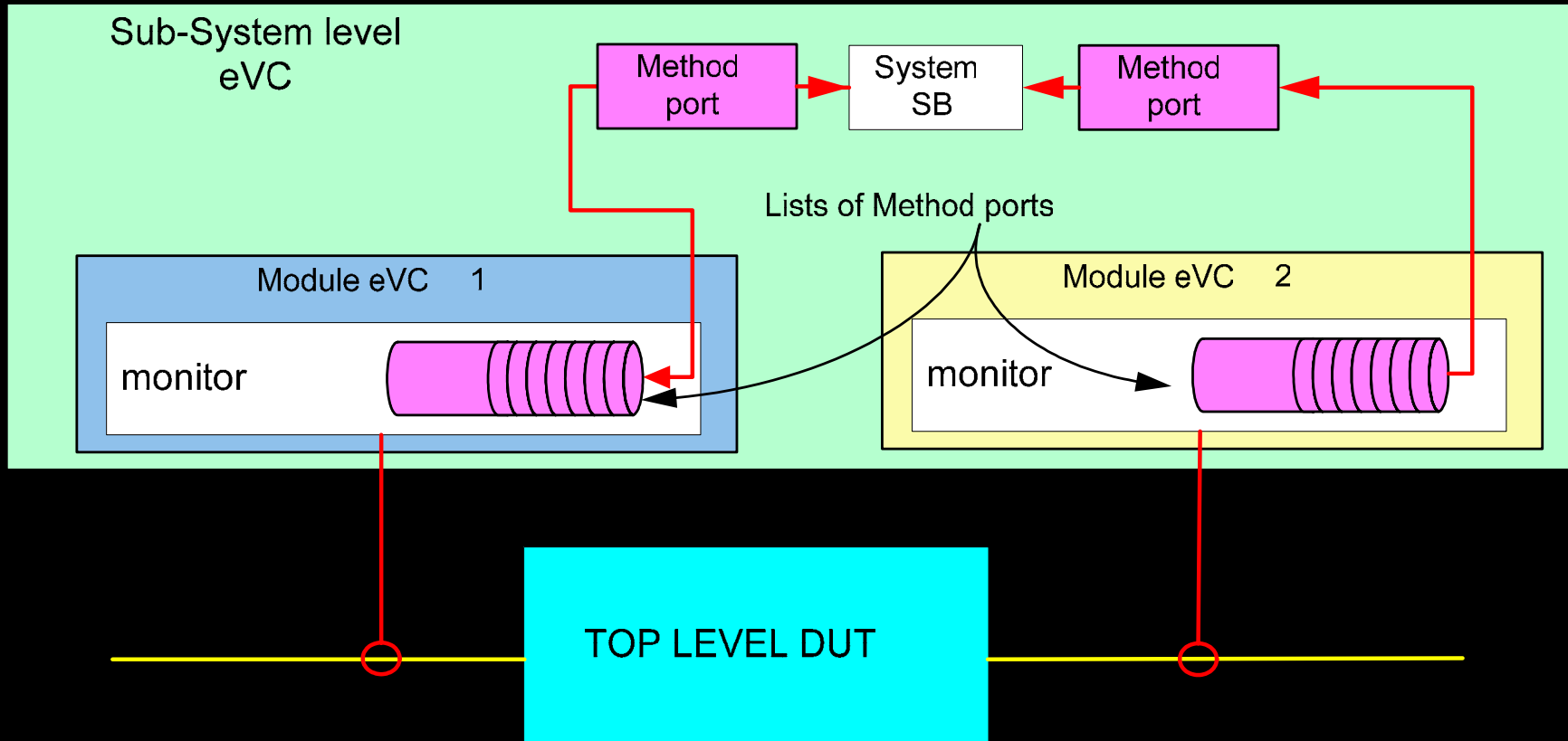
# How does it work

- Method ports essentially allow methods to be treated like data types.
- Method ports can be pushed onto lists
- Here, the key point is a list of method port at the source of the data extraction, and ports to be pushed onto it where the data is to be used
- This allows other eVCs/Units to get copies of the extracted data by adding requestor method port to source list

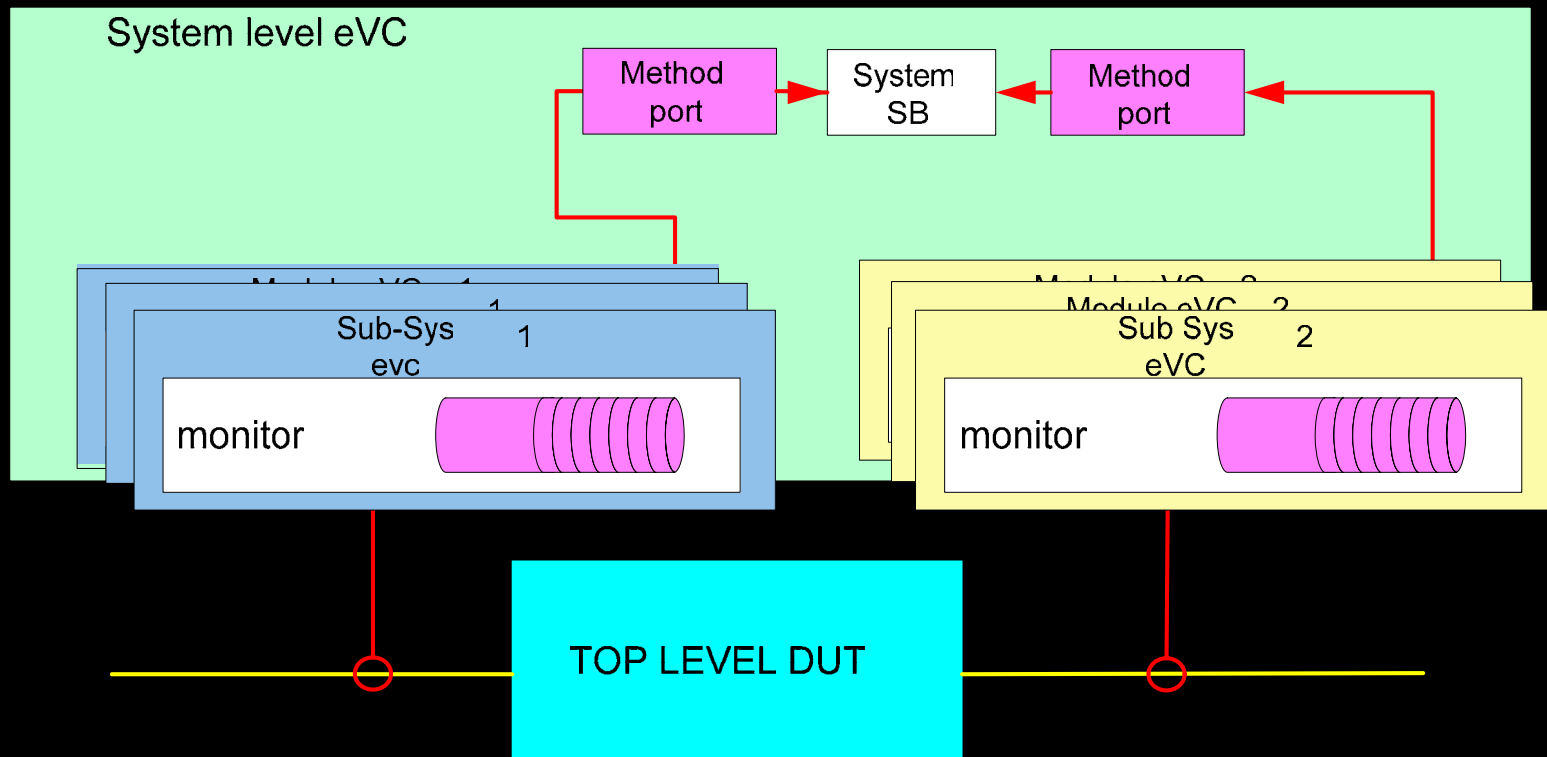
# A Sub-system level verification environment



# Sub-System level Scoreboard connection using Method ports



# Sub-System level to System level



# Advantages

- Easy to connect module level scoreboard to the system level scoreboard
- Easy to enable/disable various scoreboards.
- Easy to exchange the data from one monitor of eVCs to other eVCs
- Avoid duplication of data extractor monitor by populating information of one eVCs to other eVCs seamlessly

# How to implement (monitor side)

- Define a method port type that matches the prototype of the associated method
  - `method_type sys_sb_message_t (message : trans_s);`
  - Here, `trans_s` is a common “struct” that should be used across many eVCs. So , it’s better to define the struct at the global define file
- Create list of out method ports at the data extractor unit (the place where data extraction happens)
  - `out_trans_l : list of out method_port of sys_sb_message_t;`

# How to implement (monitor cont.)

- Create a method to collect data by calling each method port in the list
  - `Send_msg(message : trans_s) is {  
    for each in out_trans_l {it$(message); };  
};`
- Call this method whenever the data extractor has something to report
  - `data_extract_tcm() is @clk {  
    ....  
    send_msg(cap_data_s);  
};`

# How to implement (scoreboard side)

In the scoreboard unit(place where data needs to be exchanged):

- specify output method port instance
  - `collect_trans` : out method\_port of `sys_sb_message_t` is instance;
- input method port instance
  - `collect_trans_in` : in method\_port of `sys_sb_message_t` is instance;
- Bind input to output
  - `keep bind(collect_trans, collect_trans_in);`

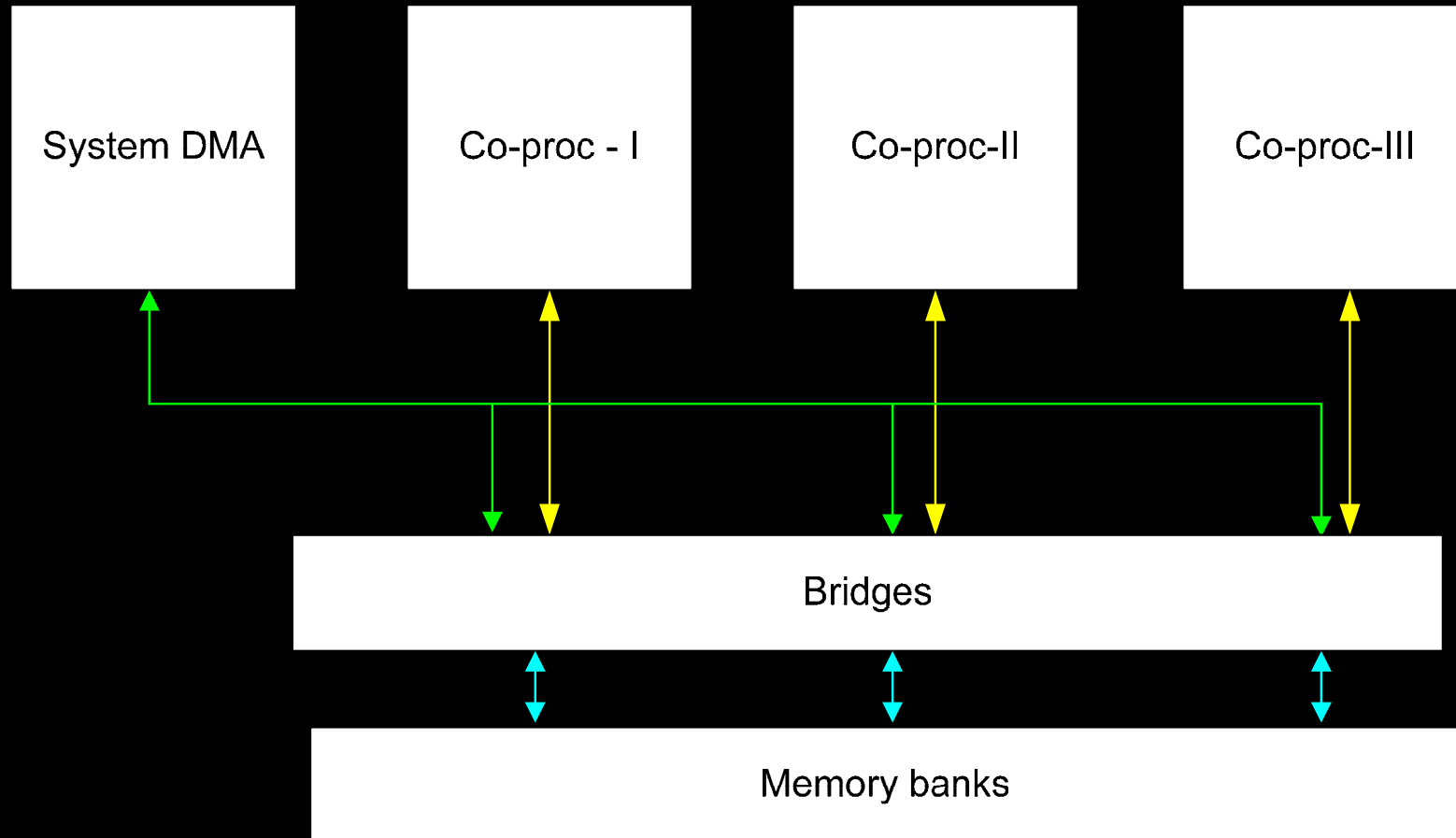
# How to implement (scoreboard cont.)

- Put call to scoreboard in “input” portion of method port
  - `collect_trans_in(info : trans_s) is {  
    call_sb(info);  
};`

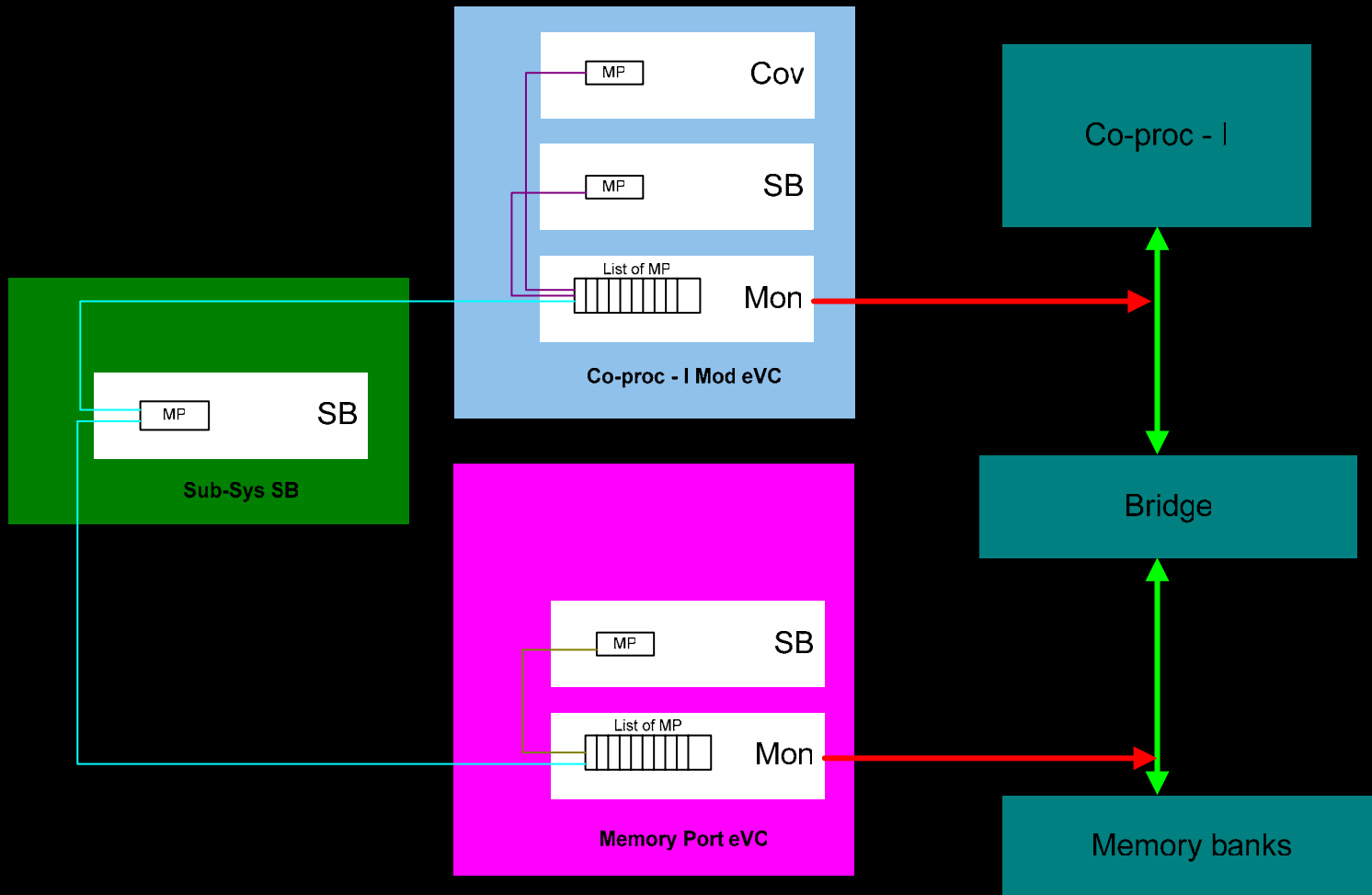
# How to implement (integration)

- Connect the scoreboard to the monitor by pushing the “input” portion of the method port onto the list in the data extractor
  - `run()` is also `{ mon.out_trans_l.add(sb.collect_trans); }`

# CASE STUDY



# CASE STUDY continues...



# CASE STUDY continues...

- Co-processor sub-system contains three co-processors and memories used by the three co-processors
- System DMA is used to load data into the memories and transfer processed data to other locations in the system
- All co-processors connected to memories via bridge
- System DMA is also connected to memories via bridge
- All co-processors are verified in the module level with their respective eVCs
- Each of the module eVCs contains data extractor, scoreboard and coverage monitor

# CASE STUDY continues...

- In the module level, the verification environment contains only module level eVC and various BFMs
- In the module eVC, Co-processor scoreboard unit and coverage monitor unit gets data from data extractor unit via method ports
- Method ports at scoreboard unit and coverage monitor unit are added to the list of method ports at data extractor unit and gets the data whenever data available
- Scoreboard at the module level is mainly intended to verify the property of the module extensively
- In the sub-system level, the main interest is verifying data path across various masters to/from slaves
- In this case, verifying data path between System DMA to memories and Co-processors to memories

# CASE STUDY continues...

- The sub-system level verification environment is constructed with co-processors module level eVCs, memory port eVCs and System DMA port eVCs
- A sub-system level scoreboard was constructed by adding the method ports of the sub-system SB to the list of method ports at various monitors
- For example, in the case of scoreboard across co-processor-1 and memories, the scoreboard was easily constructed by adding the method port at the sub-system level SB to both list of the method ports at monitor of co-processor eVC and monitor of the memory eVC
- The sub-system level SB gets data from both monitors during write and read

# CASE STUDY continues...

- When write is initiated from the co-processor, the sub-system level SB gets data from co-processor eVC's monitor and SB stores the data
- When write is seen at the memory port, the sub-system level SB gets data from memory port eVCs and compares that data against previous stored data(received from co-processor monitor)
- The same way data comparison occurs during read also
- One important point to be noted is during sub-system level verification, the module level verification environment also present and active
- Any system level application test carried out at the sub-system level in a way verifies module level properties using module level scoreboards and sub-system level properties using system level scoreboards

# Summary

- Method ports are a convenient way to implement interfaces to data extractors
- Using method ports to connect scoreboards to data extractors has several advantages:
  - Easier implementation
  - Lower risk of mistakes
  - Easier integration into system & subsystem

## Summary (cont.)

A concise example of this approach can be found in the "Method Ports" section of the "E Reference Manual".

For Specman 4.3.2, the example can be found in section 4.1.1.2.

This example was created and submitted by the authors of this paper.