



The AVM and OVM in IP Core Verification – Experiences and Observations

Gareth Edwards
IP Solutions

Topics

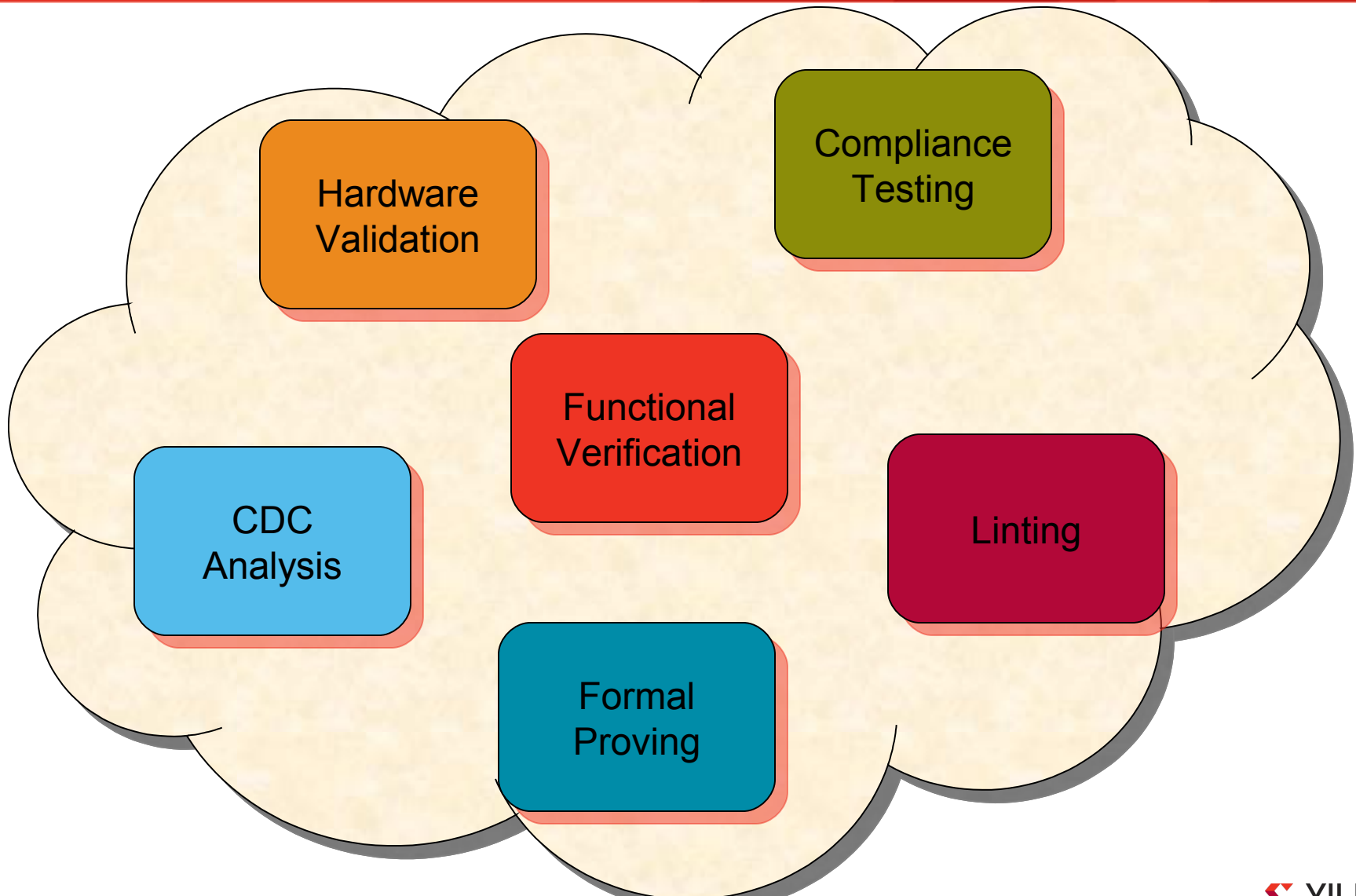
- **What is a methodology? What is it not?**
- **The timeline**
- **Some example OVM test environments**

What is methodology?

methodology

1. A collection of methods, practices, procedures and rules used by those who work in some field.
2. The study of such methods etc.
3. The implementation of such methods etc.

Methodology is only part of the overall strategy



The Timeline

- **AVM 2.0 (July 2006)**

- **AVM 3.0 (May 2007)**

- ovm_object.clone()
- Multiple environments
- Component Removal

- **OVM 1.0 (January 2008)**

- Factories
- Configuration
- Sequences/scenarios
- Backward compatible with AVM 3.0

- **OVM 1.1 (May 2008)**

- Singletop top
- Refined test phasing
- Backward compatible with OVM 1.0

- **OVM 2.0 (Sep 2008)**

- Unified sequences
- User Guide!
- Backward compatible with OVM 1.1

Various Ethernet testbenches
Wrote our own extensions
(primarily component removal)

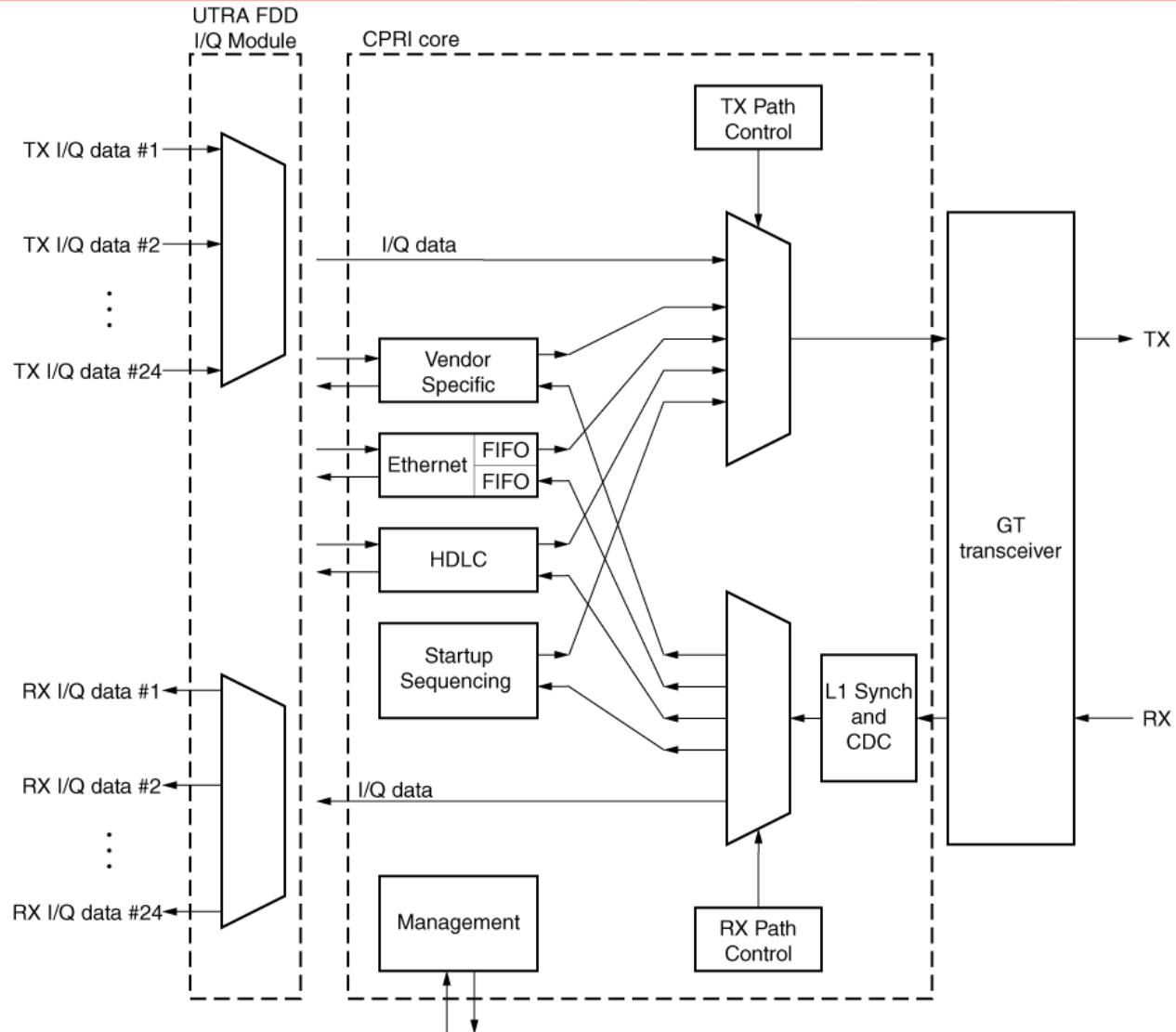
Started CPRI testbench
development, November
2007

Ported CPRI testbench in-
flight

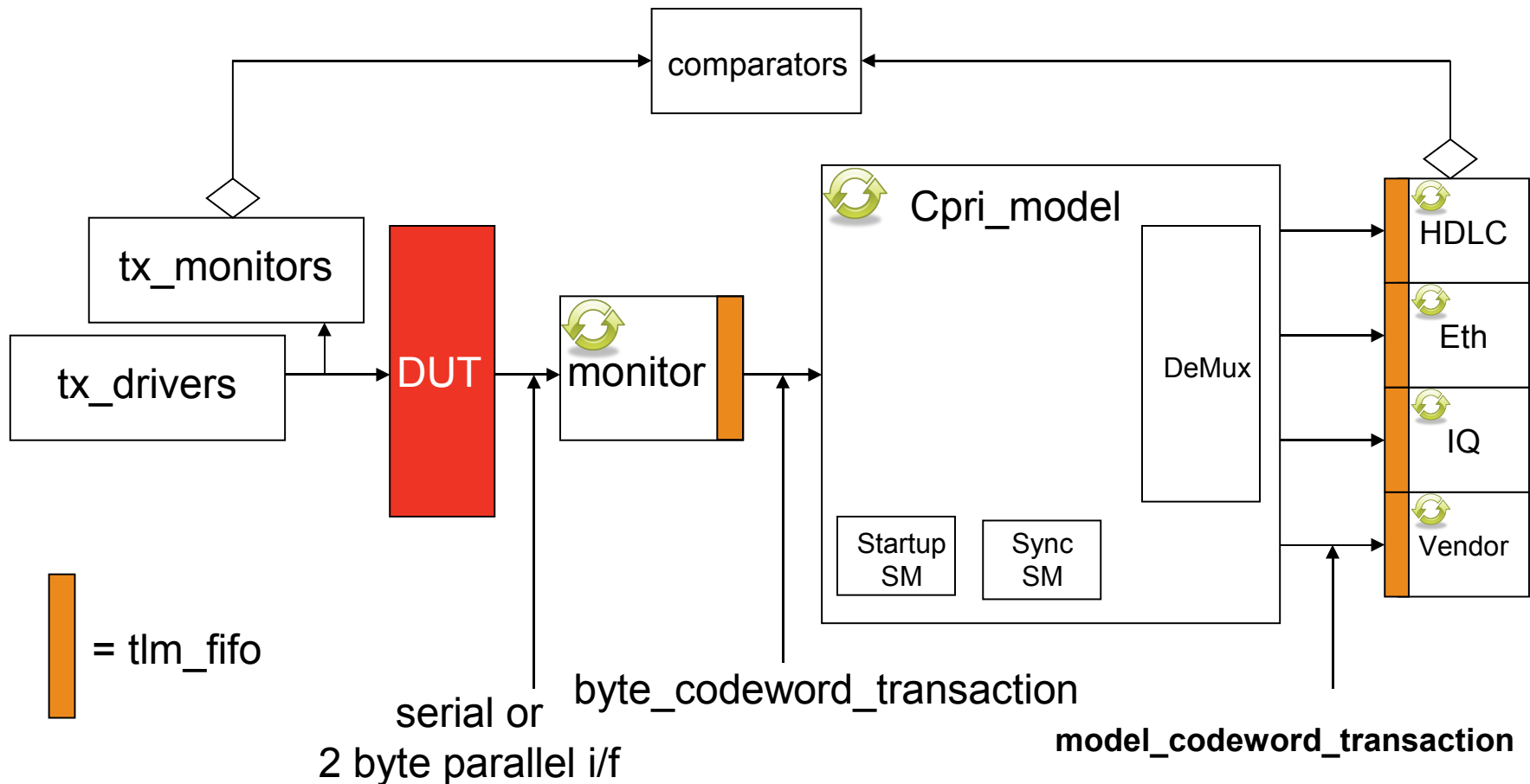
Ported CPRI testbench
again...

CPRI I/Q Module
testbenches

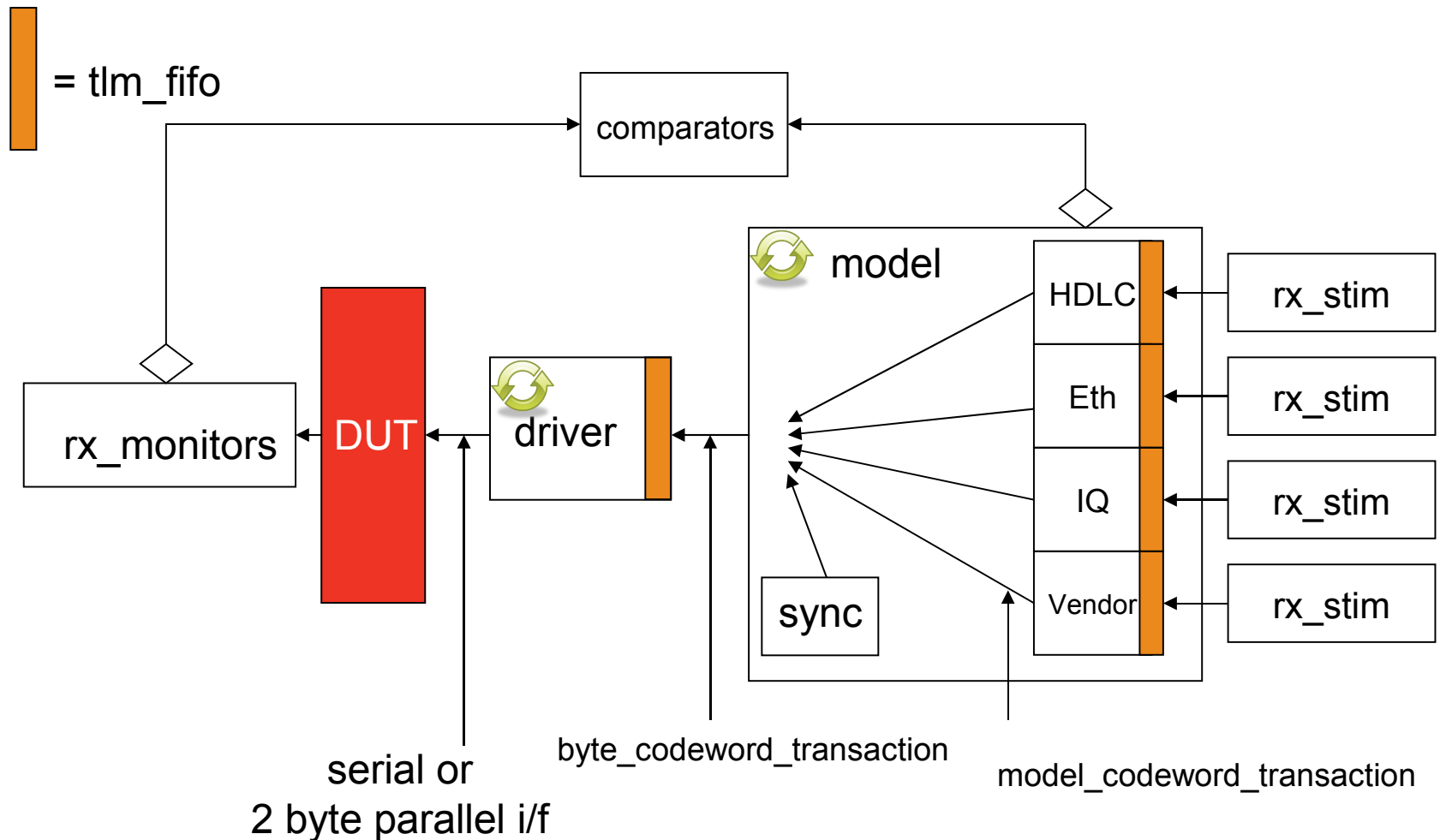
The CPRI core – the DUT



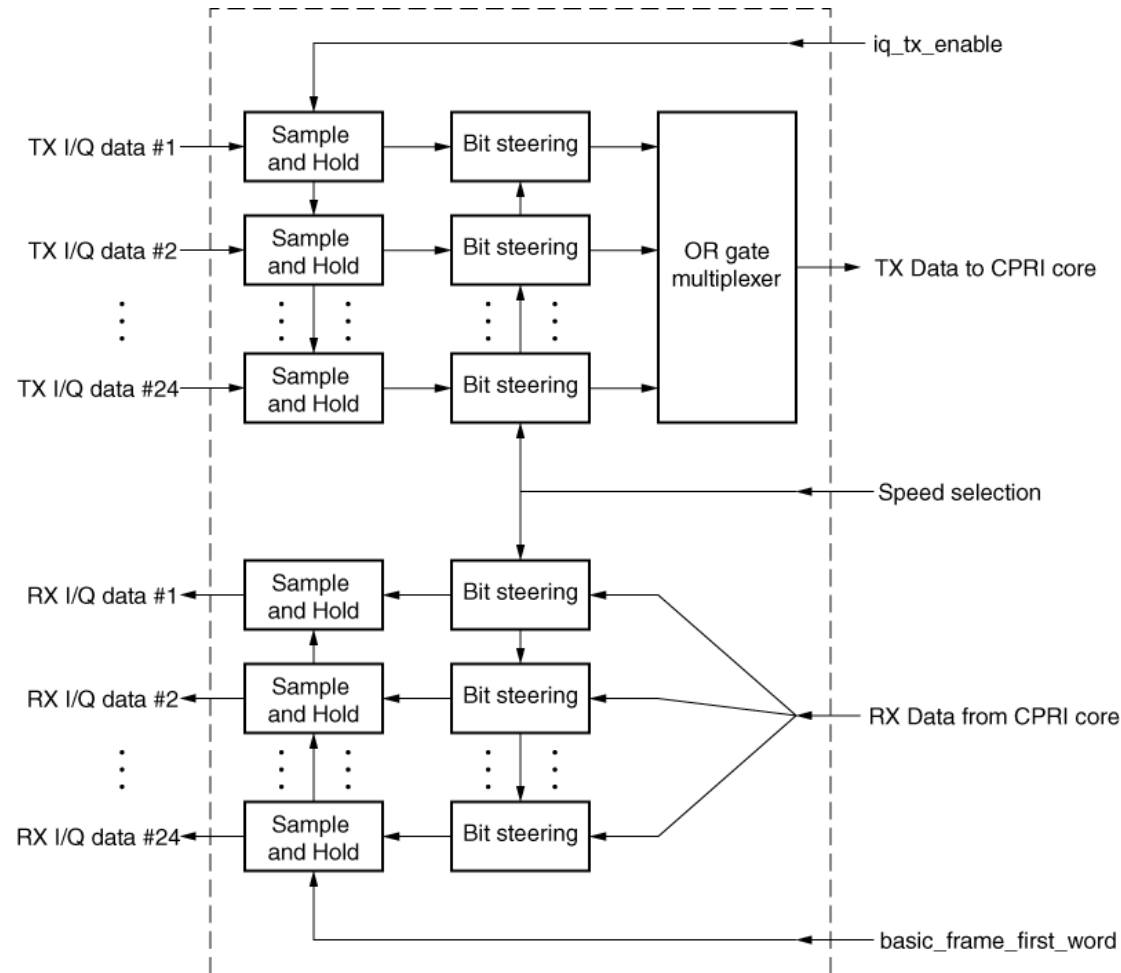
The main CPRI Test suite (transmit)



The main CPRI Test suite (receive)



The I/Q Module



```
entity iq_module is
generic (
    C_TX_WIDTH_1 : natural := 10;
    C_TX_START_1 : natural := 0;
    C_RX_WIDTH_1 : natural := 10;
    C_RX_START_1 : natural := 0;

    C_TX_WIDTH_2 : natural := 10;
    C_TX_START_2 : natural := 20;
    C_RX_WIDTH_2 : natural := 10;
    C_RX_START_2 : natural := 20;

    C_TX_WIDTH_3 : natural := 10;
    C_TX_START_3 : natural := 40;
    C_RX_WIDTH_3 : natural := 10;
    C_RX_START_3 : natural := 40;
    ...

```

Testing the I/Q Module – parameterisation (1)

```
class dut_param extends ovm_object;

  rand CpriSpeedType min_speed;

  rand int n_tx_channels;
  int tx_width[1:24];
  int tx_start[1:24];

  rand int n_rx_channels;
  int rx_width[1:24];
  int rx_start[1:24];

  function void pre_randomize();
    int status;
    status = std::randomize(tx_width, rx_width)
    with {
      foreach (tx_width[i]) {
        tx_width[i] inside {[4:20]};
      }
      foreach (rx_width[i]) {
        rx_width[i] inside {[4:20]};
      }
    };
  endfunction : pre_randomize
```

```
entity iq_module is
  generic (
    C_TX_WIDTH_1 : natural := 10;
    C_TX_START_1 : natural := 0;
    C_RX_WIDTH_1 : natural := 10;
    C_RX_START_1 : natural := 0;

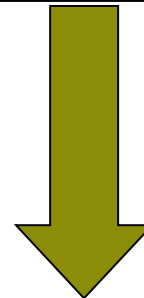
    C_TX_WIDTH_2 : natural := 10;
    C_TX_START_2 : natural := 20;
    C_RX_WIDTH_2 : natural := 10;
    C_RX_START_2 : natural := 20;

    C_TX_WIDTH_3 : natural := 10;
    C_TX_START_3 : natural := 40;
    C_RX_WIDTH_3 : natural := 10;
    C_RX_START_3 : natural := 40;
    ...
  )
```

Testing the I/Q Module – parameterisation (2)

```
class dut_param extends ovm_object;  
  
    rand CpriSpeedType min_speed;  
  
    rand int n_tx_channels;  
    int tx_width[1:24];  
    int tx_start[1:24];  
  
    rand int n_rx_channels;  
    int rx_width[1:24];  
    int rx_start[1:24];  
  
    function void pre_randomize();  
        int status;  
        status = std::randomize(tx_width, rx_width)  
        with {  
            foreach (tx_width[i]) {  
                tx_width[i] inside {[4:20]};  
            }  
            foreach (rx_width[i]) {  
                rx_width[i] inside {[4:20]};  
            }  
        };  
    endfunction : pre_randomize
```

Stub program creates
dut_param object,
randomizes and saves
generics to storage



Simulator reads generics,
elaborates testbench
and reconstructs dut_param
object

Functional Coverage Tracking

X GamePlan (TM) Verification Planner 1.2 by Jasper Design Automation - /Sandbox/kgarden/IPsandbox/groups/cni/mb_emac_sv_v1_0/test/mb_emac

File Edit View Integration Reports Help

CREATE ANALYZE

Plan Elements

- PL1: Mt. Blanc Hard TEMAC
 - PL1.F1: Client Interface
 - PL1.F1.F1: TX
 - PL1.F1.F1.F1: Client Signal Checks
 - PL1.F1.F1.F1.F1: Underrun position
 - PL1.F1.F1.F1.F1.P1: Underrun before Acknowledge
 - PL1.F1.F1.F1.F1.P2: Data Valid held high after underrun
 - PL1.F1.F1.F1.F1.P3: Data Valid deasserted after underrun
 - PL1.F1.F1.F1.F1.P4: Underrun after Data Valid
 - PL1.F1.F1.F1.F2: Data Valid spacing
 - PL1.F1.F1.F1.F2.P1: Random
 - PL1.F1.F1.F1.F2.P2: One Cycle
 - PL1.F1.F1.F1.F3: Pause Request tests
 - PL1.F1.F1.F1.F3.P1: Request asserted during frame
 - PL1.F1.F1.F1.F3.P2: Request asserted during IFG
 - PL1.F1.F1.F1.F3.P3: Pause Address
 - PL1.F1.F1.F1.F3.P4: Pause Value
 - PL1.F1.F1.F1.F4: Statistics Signal Checks
 - PL1.F1.F1.F1.F4.P1: Stats Valid
 - PL1.F1.F1.F1.F4.P2: Byte Valid
 - PL1.F1.F1.F1.P1: TX Frame Acknowledged
 - PL1.F1.F1.F1.P2: TX Frame Acknowledged only once
 - PL1.F1.F1.F1.P3: Retransmit asserted with Collision
 - PL1.F1.F1.F1.P4: Interframe Gap Tests
 - PL1.F1.F2: RX
 - PL1.F1.F2.F1: Client Signal Checks
 - PL1.F1.F2.F1.F1: Statistics Signal Checks
 - PL1.F1.F2.F1.F1.P1: Stats Valid
 - PL1.F1.F2.F1.F1.P2: Byte Valid
 - PL1.F1.F2.F1.P1: Good/Bad Frame Exclusivity
 - PL1.F1.F2.F1.P2: Good/Bad Frame Assertion
 - PL1.F2.F1: Physical Interface
 - PL1.F2.F1.F1: General PHY Behaviour
 - PL1.F2.F1.F1.F1: TX
 - PL1.F2.F1.F1.F1.P1: Collision Status
 - PL1.F2.F1.F1.F1.P2: Carrier Sense Status
 - PL1.F2.F1.F1.F1.P3: Error Status
 - PL1.F2.F1.F1.F1.P4: Transmission of Extension codes
 - PL1.F2.F1.F1.F1.P5: Transmit Preamble Check
 - PL1.F2.F1.F1.F1.P6: 1/3*IFG Rule
 - PL1.F2.F1.F1.F1.P7: Minimum IFG Check
 - PL1.F2.F1.F2: RX
 - PL1.F2.F1.F2.P1: Preamble Checks

Element Details

Description | Attributes | Results

Number
PL1.F1.F1.F1.P3

Name
Retransmit asserted with Collision

Verification Management

Section	Testplan Section / Coverage Link	Coverage	CoverageGraph	Goal	Description
0	testplan	98.179%		95%	
PL1	Mt. Blanc Hard TEMAC	98.179%		95%	Relevant Specifications: (a) IEEE 802.3-...
PL1.F1	Client Interface	96.25%		95%	
PL1.F1...	TX	92.5%		100%	
PL1.F1...	RX	100%		100%	
PL1.F1...	Client Signal Checks	100%		100%	
PL1.F1...	Statistics Signal Checks	100%		100%	
PL1.F1...	Stats Valid	100%		100%	6.1.4 Check Stats valid is high for exactly...
PL1.F1...	/tb_top/client_rx_bind/ap_stats_valid	100%		100%	
PL1.F1...	Byte Valid	100%		100%	6.1.5 Stats_byte_valid should be high for ...
PL1.F1...	IGood/Bad Frame Exclusivity\	100%		100%	6.1.8 GOODFRAME and BADFRAME ca...
PL1.F1...	IGood/Bad Frame Assertion\	100%		100%	Good frame or bad frame must go high so...
PL1.F2	Physical Interface	97.9167%		95%	
PL1.F2...	General PHY Behaviour	95.83333%		100%	
PL1.F2...	Protocol-Specific PHY Behaviour	100%		100%	
PL1.F2...	GMII/MII	100%		100%	
PL1.F2...	TX	100%		100%	
PL1.F2...	RX	100%		100%	
PL1.F2...	RX Valid Checks	100%		100%	bins early // 3.2.2 One cycle early bins la...
PL1.F2...	RX Error Checks	100%		100%	bins during_frame // 3.2.6 bins between...
PL1.F2...	RGMI	100%		100%	
PL1.F2...	Serial	100%		100%	
PL1.F3	MAC Functionality	100%		95%	
PL1.F4	Ethernet Frame	100%		95%	
PL1.F5	Configuration	100%		95%	
PL1.F6	Management	94.9074%		95%	

Find: GO

Conclusions

- **Using a Functional Verification Methodology can improve the quality of your verification effort**
- **We've used OVM; other verification methodologies are available**
- **It's not that important which one you use**
- **Don't forget the rest of your verification strategy**

Questions?

