



Open Verification Methodology

Beyond a common base class library: reduce work by reusing OVM agents on common interfaces

Stacey Secatch
Senior Staff Verification Engineer

June 15th, 2010

© Copyright 2010 Xilinx

Agenda

- **Special requirements for FPGA IP**
- **Xilinx Serial RapidIO core basics**
- **Reuse within the bus architecture**
- **Using configuration objects to customize components**
- **Summary**

FPGA IP is different

- **Highly parameterizable source code**

- And all possible permutations must be tested
 - In practice, follow constrained random methodology for parameters
- “Ship” RTL, customized netlist delivered on-the-fly

- **Usable in many different ways**

- No single golden system model
- Every allowable user environment must be tested

- **Development cycle**

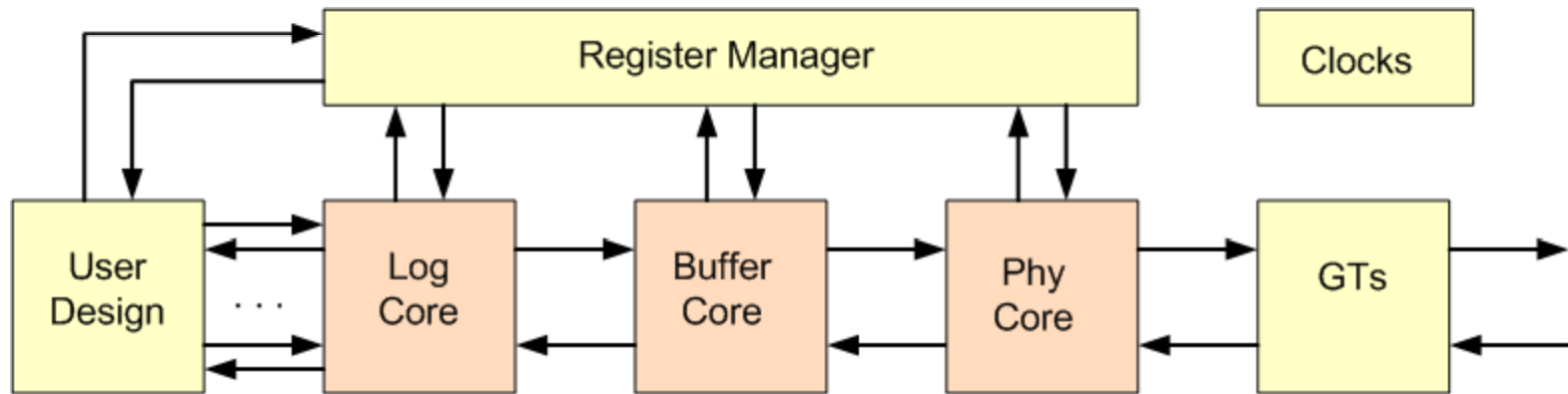
- Short design cycle
- Multiple releases for any given IP
 - Feature updates
 - Performance enhancements
 - Bug fixes



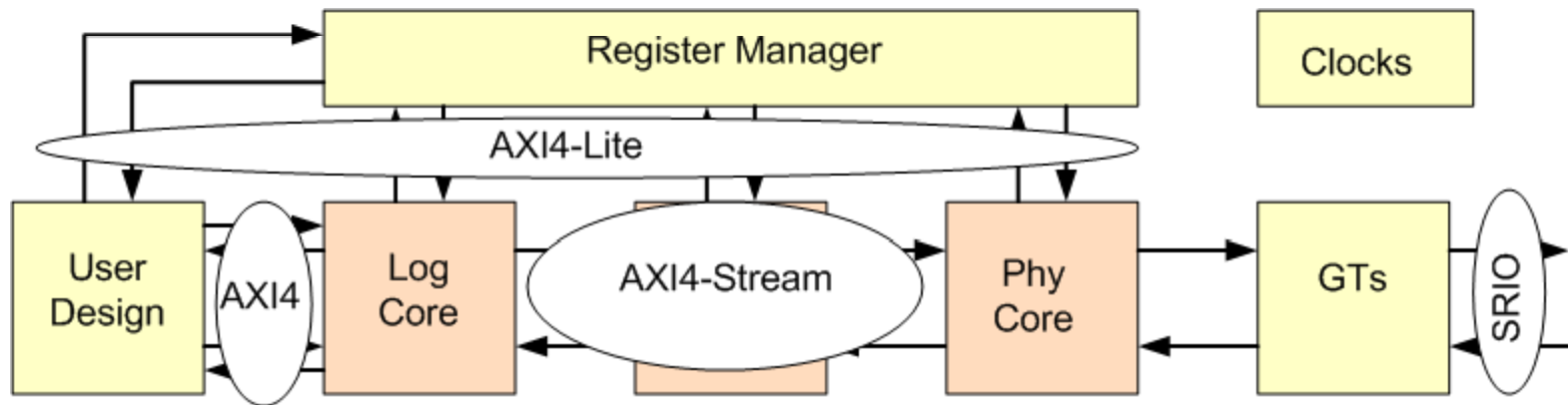
Serial RapidIO protocol basics

- **“Lightweight” embedded connectivity protocol**
 - Still a layered protocol
- **Connects with a serial transceiver**
 - Link partners negotiate connection based on feature support
- **Packets are transmitted over the link**
 - Flushed when acknowledged by link partner
 - Automatically retransmitted when there is a packet error
 - Priority based throttling of throughput can be configured by either link partner
 - Reordering of packets is allowed

Serial RapidIO core architecture



Serial RapidIO core architecture

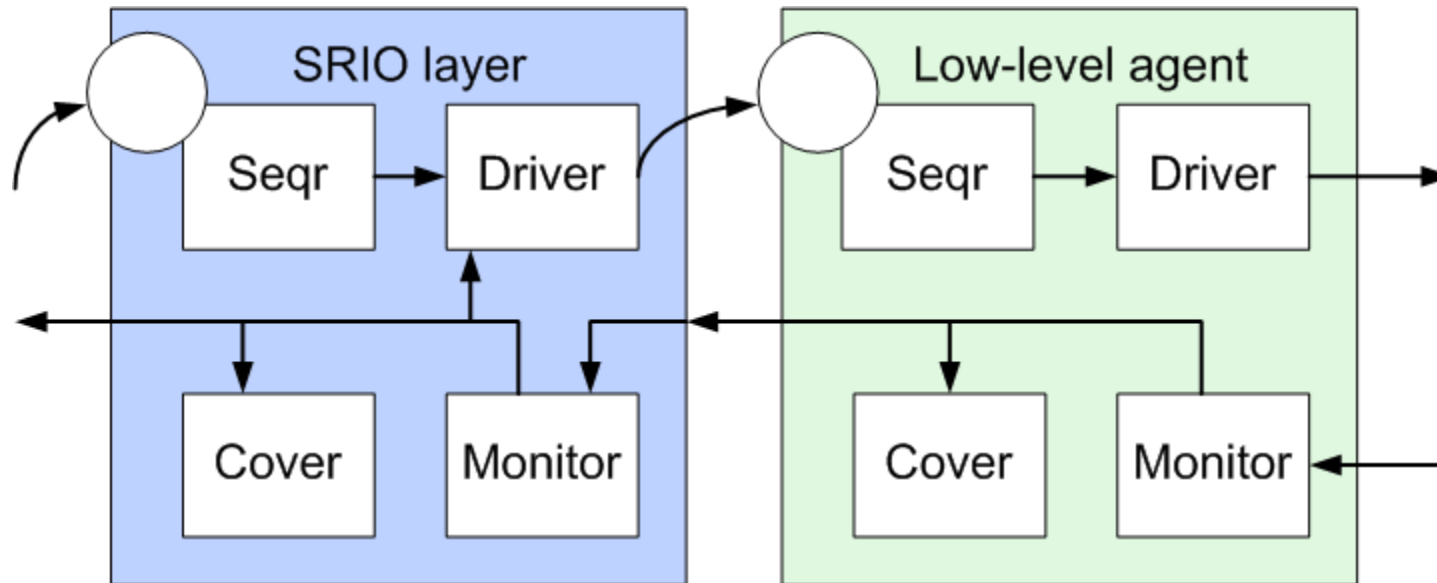


Serial RapidIO on top of AXI4-Stream

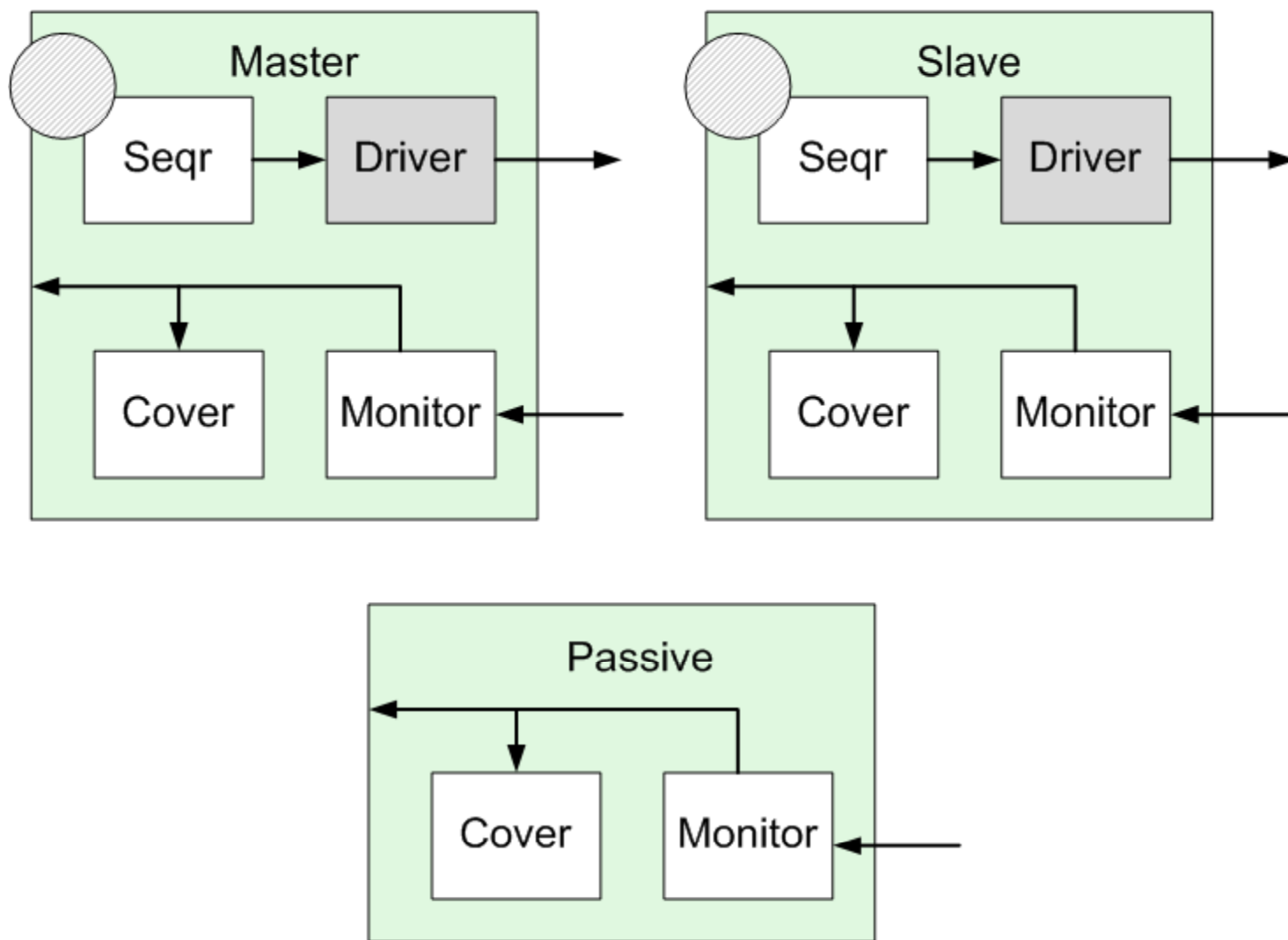
- **AXI4-Stream controls data flow**
 - Handshaking on either side of interface (ie, ready/valid)
 - Packet delimiting
 - User signal for addition control (ie, discontinues)
- **SRIO packets are stuffed into AXI4-Stream**
 - All of SRIO packet becomes data field
 - User signal must be set appropriately
 - Streaming handshaking all handled by low level



Generic SRIO AXI4-Stream environment architecture



Reuse among bus types

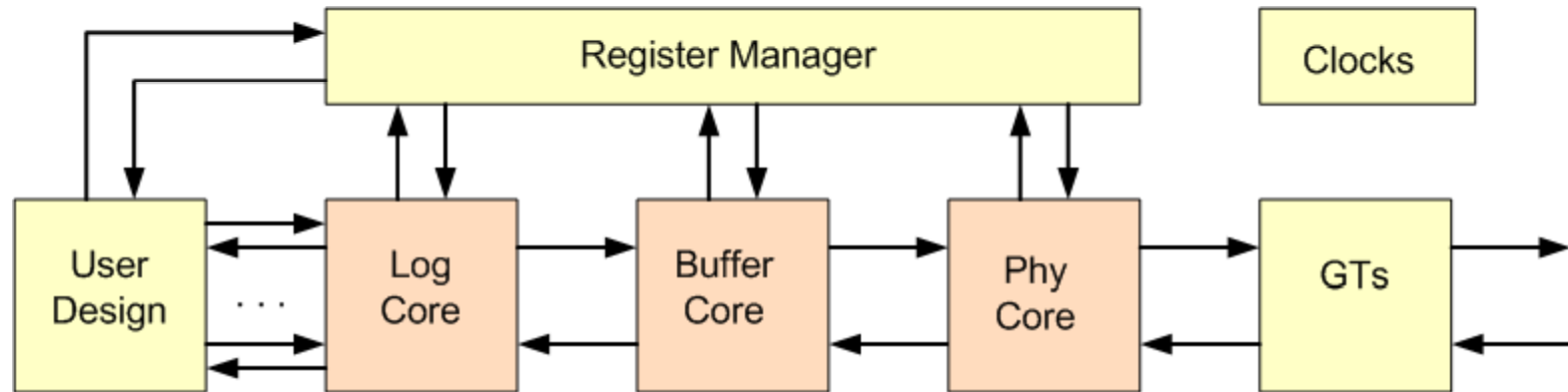


Determining usage of component

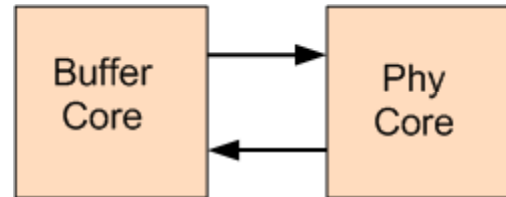
- Use configuration space along with polymorphism to determine how the component is being used:

```
xas_driver_base `PARAMS_MAP driver;  
  
void'(get_config_int("is_mst_active", is_mst_active));  
void'(get_config_int("is_slv_active", is_slv_active));  
  
if (is_mst_active == OVM_ACTIVE) begin  
    driver = xas_mst_driver `PARAMS_MAP::  
                type_id::create("driver", this);  
end else if (is_slv_active == OVM_ACTIVE) begin  
    driver = xas_slv_driver `PARAMS_MAP::  
                type_id::create("driver", this);  
end
```

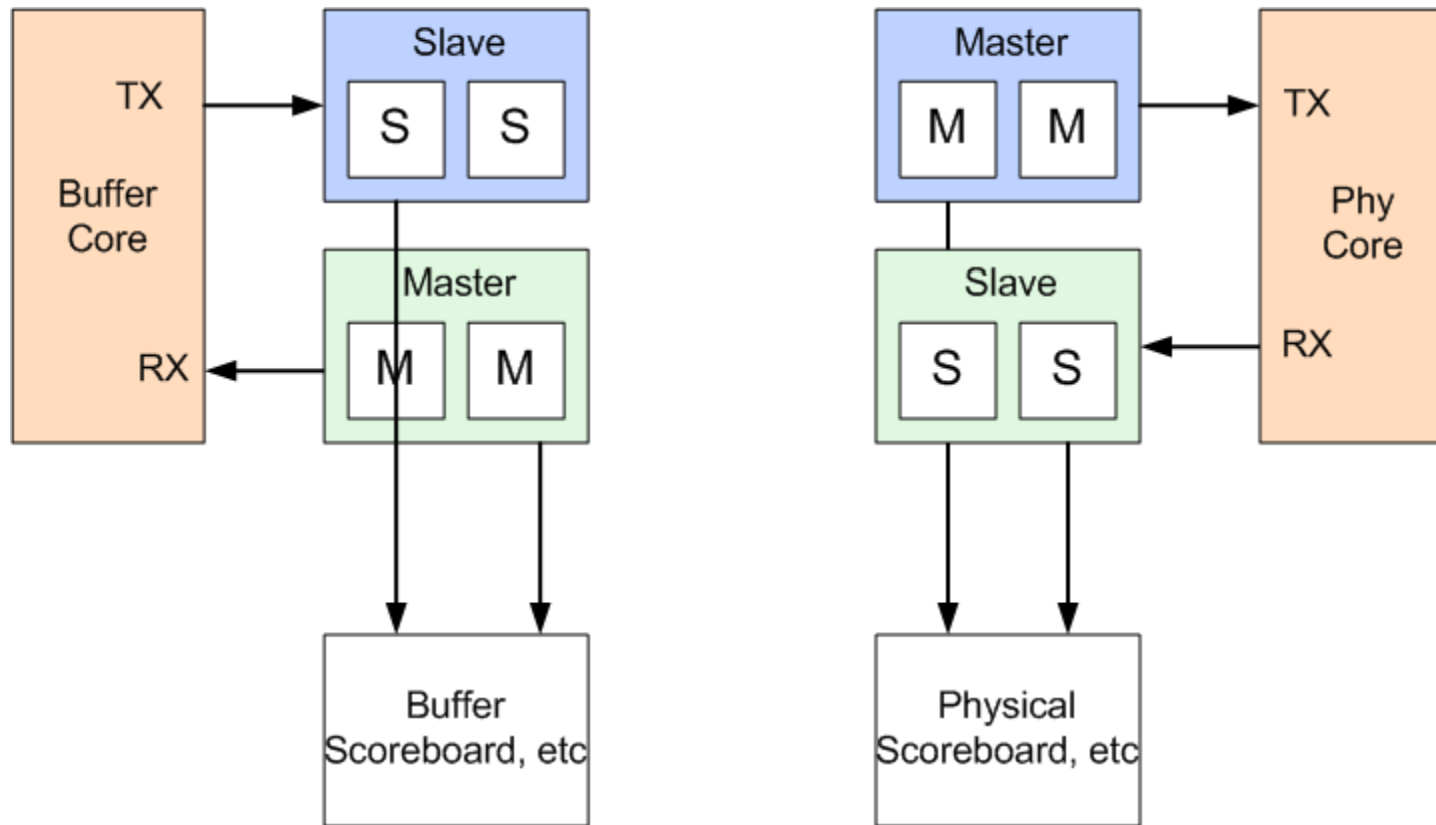
Serial RapidIO core architecture



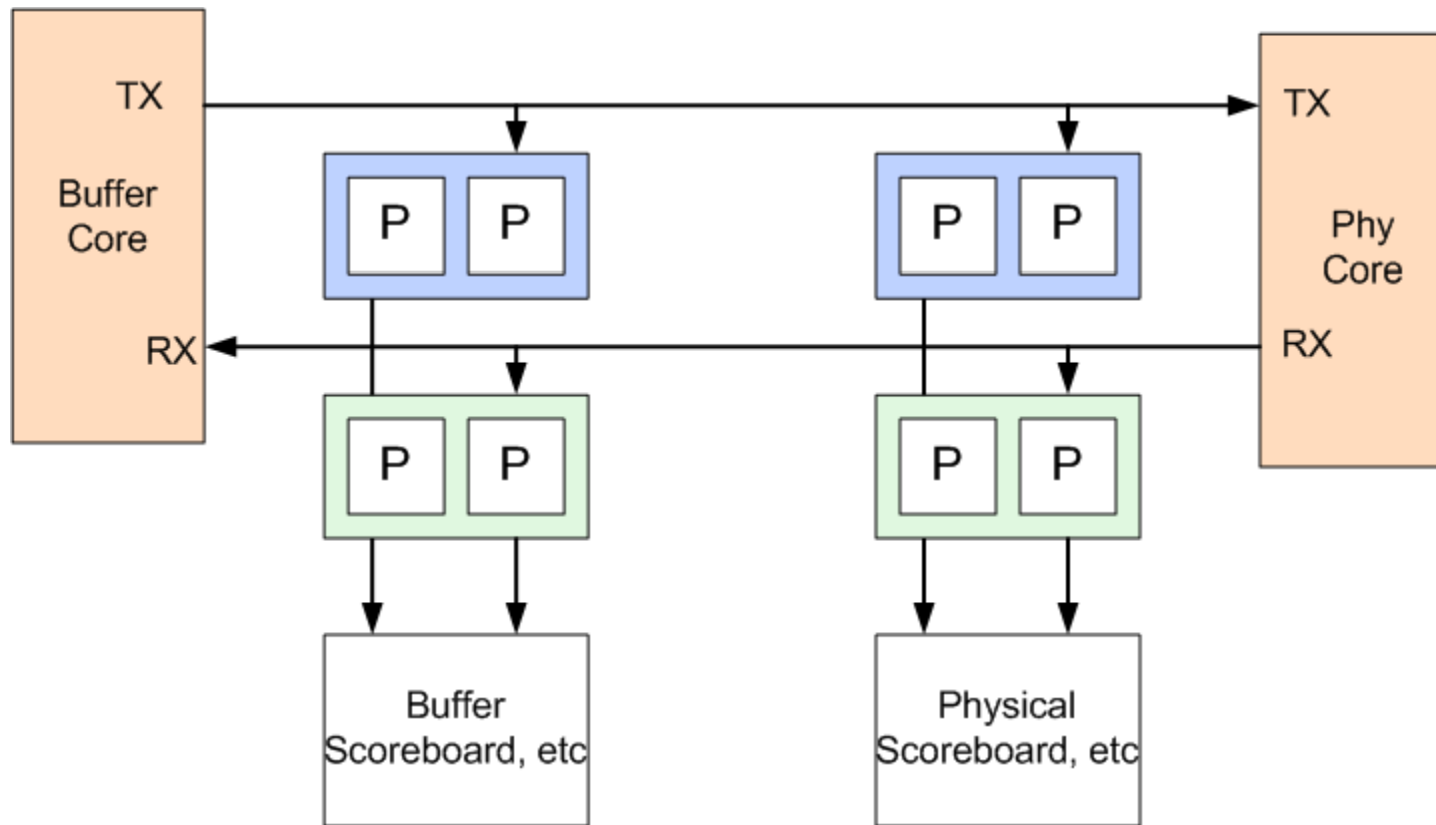
Serial RapidIO core architecture



Interface in stand-alone testbenches

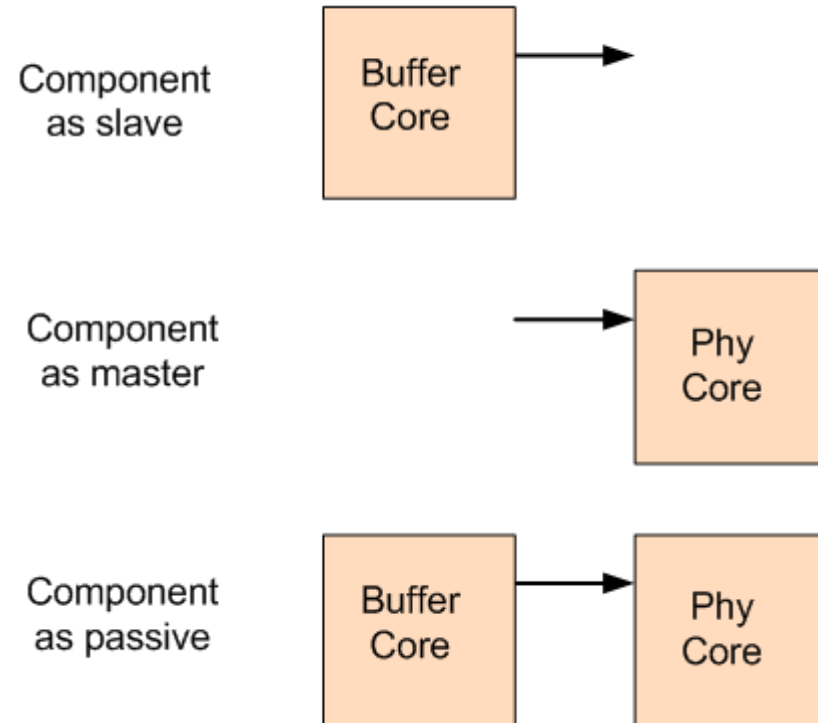


Interface when in the larger environment



Need to customize behavior for:

- What the driver forces on the interface
- What the monitor ensures happens
- What the coverage collector measures



Configuring one of the interfaces

- In the environment containing the component, set up the configuration object (depending on our configuration!):

```
if (!tb_cfg.is_buf_active) begin
  if (tb_cfg.is_phy_active) begin
    set_config_int("axis_*", "is_slv_active", OVM_PASSIVE);
    set_config_int("axis_*", "is_mst_active", OVM_ACTIVE);
    ...
    axis_cfg.valid_pkt_deassert_okay = 1;
    ...

set_config_object("axis_*", "xas_cfg", axis_cfg, 0);
axis_agnt = xas_agent #(DW, UW, IW, EW)::
    type_id::create("axis_agnt", this);
```

- In the component, get the object, and use values to control behavior:

```
if (cfg.valid_pkt_deassert_okay) begin
  ...
```

Summary

- **Work upfront for reuse pays off in reduced schedule and simplified maintenance**
- **The configuration space is a very powerful aspect of OVM**
 - Use to control usage of component as well as configuration of low level
 - But, be careful!
 - Establish naming conventions up-front to avoid namespace confusion
 - Establish methodology for providers and consumers to avoid variable conflicts
 - Determine where things get set, and how pattern matching works