DVCon 2012

Design & Verification Conference & Exhibition

February 28 – March 1, 2012

Relieving the Parameterized Coverage Headache

by

Christine Lovett

Senior Verification and Design Engineer

Xilinx

Agenda

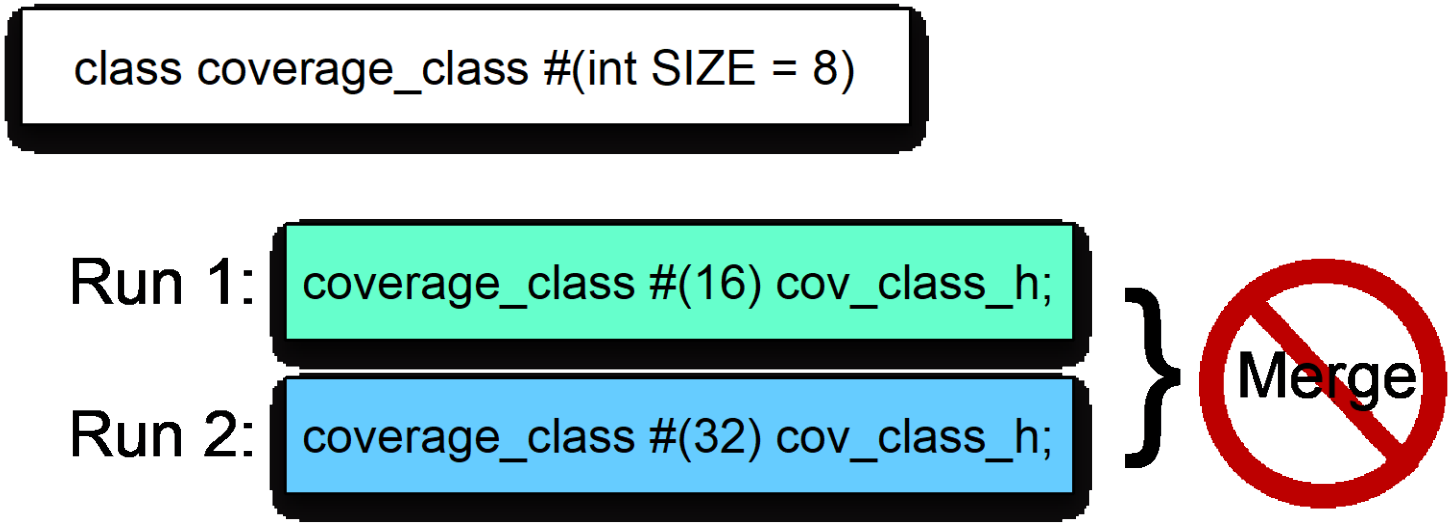
- Special requirements for customizable IP
- Difficulties with parameterized coverage
- Writing Coverage
  - Class coverage
  - Module coverage
  - Property coverage
  - Using generates
  - Crossing parameter coverage
- Summary

Customizable IP is different

- Highly parameterizable source code
  - "Ship" complete RTL within our tools after customization
  - All valid permutations must be tested
    - Constrained-random/directed methodology can be used
- Usable in many different ways
  - No single golden system model
  - Each customer ends up with customized netlist which is delivered on-the-fly

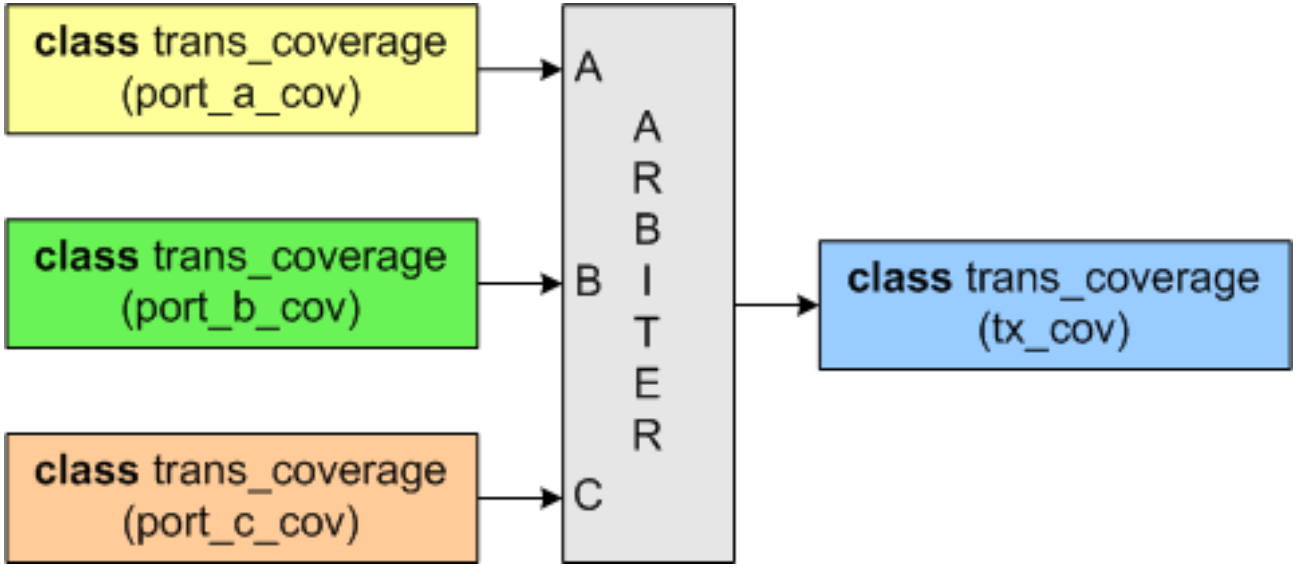
Merging Limitations

- The LRM does not indicate what should happen on a merge
- **You** need to determine what should be merged or not merged
- Each specialization of a class is its own type and won't merge as part of a basic flow



Class coverage overview

- Ideal for protocol coverage
  - Typically doesn't use parameters
- Easily reusable
- If parameters must be used consider:
  - All possible values
  - When it's valid to sample

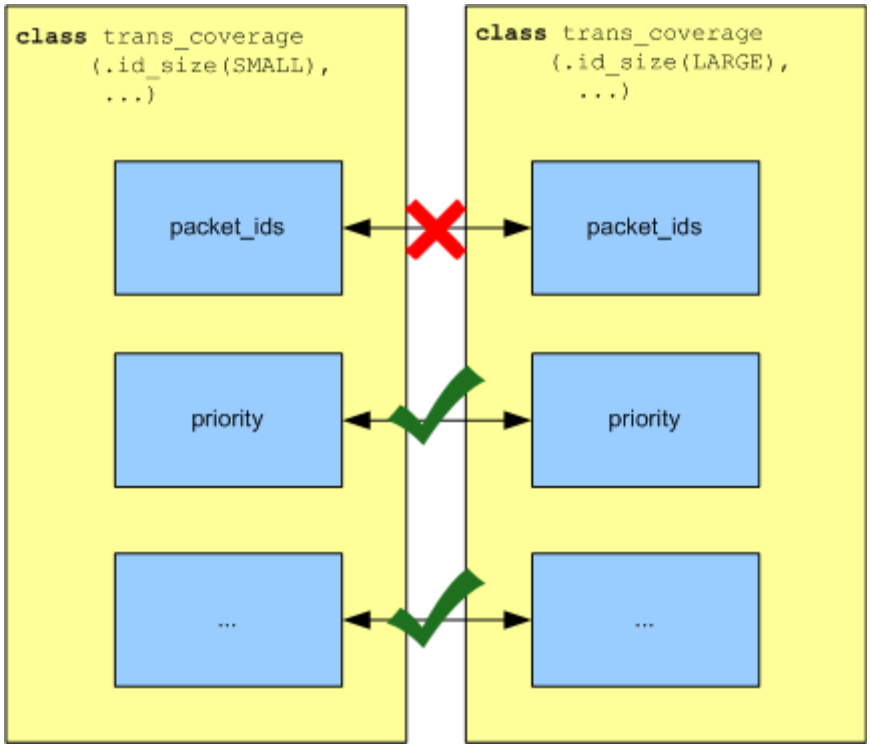


Class coverage sample code

```
class trans_coverage ...
...
  // Covering packet ID values within a class
  // when packet_ids is a parameterized width
  covergroup cg_packet_ids;
    cp_small_pid: coverpoint (packet_ids)
                              iff (id_size == small) {
      bins min = {0};
      bins mid = {[1:30]};
      bins max = {31};
    }
    cp_large_pid: coverpoint (packet_ids)
                              iff (id_size == large) {
      bins min = {0};
      bins mid = {[1:62]};
      bins max = {63};
    }
  endgroup
...
```

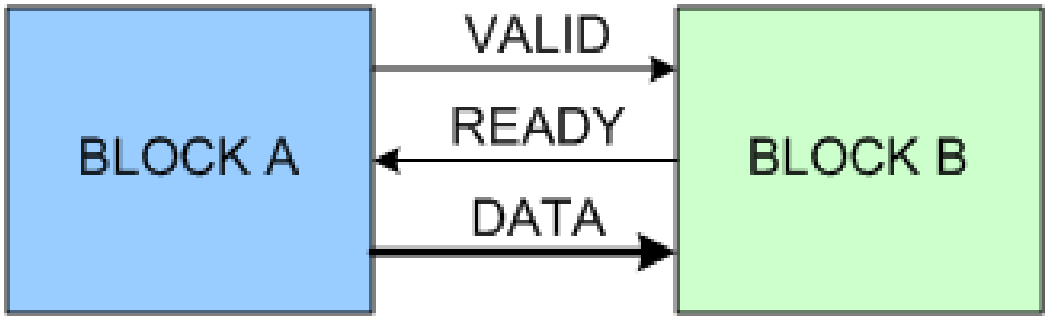
To merge or not to merge…

- We don't want packet_id coverage to merge across the ID_SIZE parameter
  - Merge across the others
- Other covergroups might depend on a different parameter
- Some might not depend on any



Module coverage overview

- Module coverage exists in:
  - Bind modules
  - RTL modules
  - SystemVerilog Interfaces
- Ideal for sub-module coverage un-seen by the high level system



- *Can take advantage of direct parameter access and the **generate** keyword*

Class coverage review

```
class trans_coverage ...
...
  // Covering packet ID values within a class
  // when packet_ids is a parameterized width
  covergroup cg_packet_ids;
    cp_small_pid: coverpoint (packet_ids)
                              iff (id_size == small) {
      bins min = {0};
      bins mid = {[1:30]};
      bins max = {31};
    }
    cp_large_pid: coverpoint (packet_ids)
                              iff (id_size == large) {
      bins min = {0};
      bins mid = {[1:62]};
      bins max = {63};
    }
  endgroup
...
```

Module coverage sample code

```
...
// Covering packet ID values within a module when
// packet_ids is a parameterized width
generate if (ID_SIZE == SMALL) begin
  covergroup cg_packet_ids;
    cp_small_pid: coverpoint (packet_ids){
      bins min = {0};
      bins mid = {[1:30]};
      bins max = {31};
    }
  endgroup
end else if (ID_SIZE == LARGE) begin
  covergroup cg_packet_ids;
    cp_large_pid: coverpoint (packet_ids){
      bins min = {0};
      bins mid = {[1:62]};
      bins max = {63};
    }
  endgroup
end endgenerate
...
```

Generates for per bit coverage

- Generates create implicit hierarchy that cannot merge
- Enables per bit coverage to be written once and reused across all bits

```
// Based on buffer depth, see each
// buffer location as full and empty
reg [BUF_DEPTH-1:0] free_locations;
...
generate
  for (int ii=0; ii < BUF_DEPTH; ii++) begin: cg_location_gen
    covergroup cg_location;
      coverpoint (free_locations[ii]);
    endgroup
  end
endgenerate
```

Generates for multi-dimensional array coverage

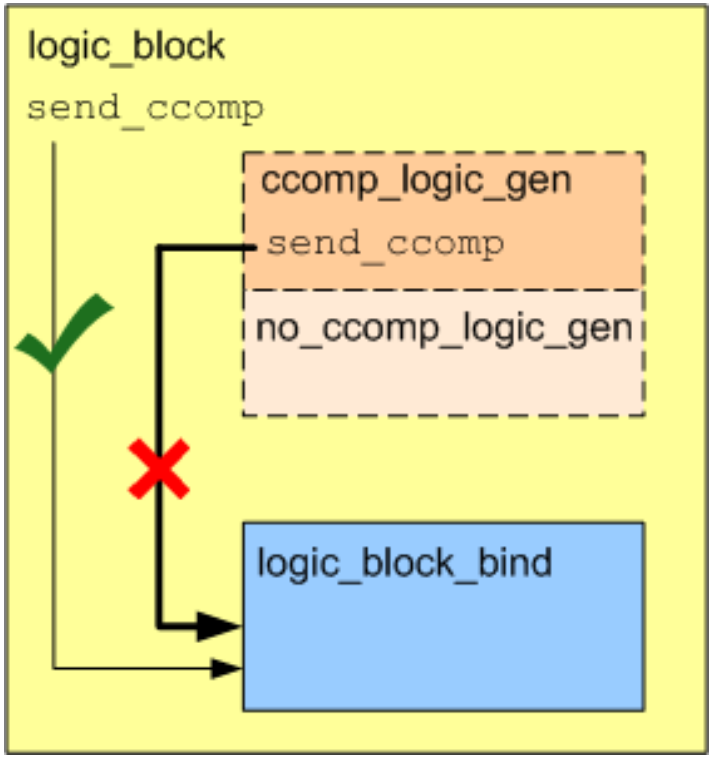
- Implicit hierarchy also eases the creation of multi-dimensional array coverage
- Creates only valid coverage depending on parameterization

```
// Check the counter reaches its max
// value on each lane.
reg [3:0] error_cnt [LINK_WIDTH-1:0];
...
generate
  for (int ii=0; ii < LINK_WIDTH; ii++) begin: error_cov_gen
    cover property
      (@(posedge clk) error_cnt[ii] == MAX_ERRORS));
  end
endgenerate
```

Property coverage background

- *Bind files can't see generated signals!*

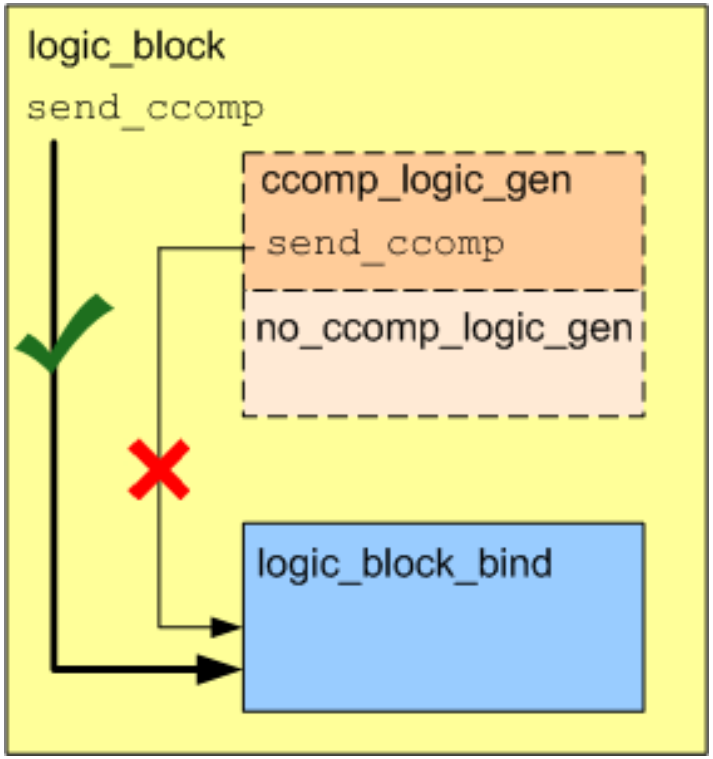
```
// A clock compensation is
// sent only if enabled

generate
  if (CCOMP_EN) begin: ccomp_logic_gen
    // Inaccessible to files bound
    // to this module and therefore
    // not recommended
    wire send_ccomp;

    assign send_ccomp =
           (ccomp_ctr == MAX_CCOMP_CNT);
    ...
  end
endgenerate
```



Property coverage background cont.

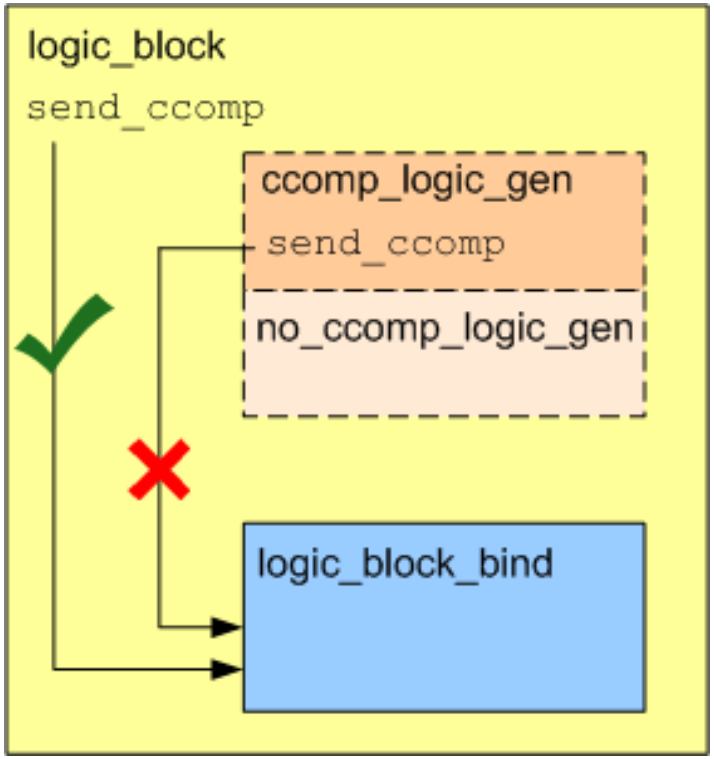
- *Bind files can't see generated signals!*

```
// A clock compensation is
// sent only if enabled
wire send_ccomp;

generate
  if (CCOMP_EN) begin: ccomp_logic_gen
    assign send_ccomp =
           (ccomp_ctr == MAX_CCOMP_CNT);
    ...
  end else begin
    // clear out when unused
    assign send_ccomp = 0;
  end
endgenerate
```



Property coverage sample code

```
// Disable properties for
// invalid configurations.
cover property
  (@(posedge clk) disable iff (!CCOMP_EN)
    (send_ccomp));

// Generate coverage only when
// valid
generate
  if (CCOMP_EN) begin: ccomp_cov_gen
    cover property
      (@(posedge clk)
        (send_ccomp));
  end
endgenerate
```



Crossing parameter coverage

- May need to cross parameters in order to get desired coverage

```
// cg_fifo_full covers the full signal for each
// valid value of FIFO_DEPTH
covergroup cg_fifo_full;
  cp_fifo_full : coverpoint (fifo_full);


  // Duplicated parameter coverage
  cp_fifo_depth: coverpoint (FIFO_DEPTH) {
    bins min = {8};
    bins max = {16};
  }

  cross cp_fifo_full, cp_fifo_depth {
    ignore_bins ignore = binsof (cp_fifo_full) intersect {0};
  }
endgroup
```

Summary

- The current state of the language doesn't support parameterized designs perfectly
  - Bug your vendor for the language enhancements!
- Most problems can be worked around, but a lot of extra code tends to be introduced
- Can use methodology to minimize the problem:
  - Keeping coverage classes non-parameterized
  - Smart use of generate statements/for loops

Appendix:

REFERENCE SLIDES

LRM Details

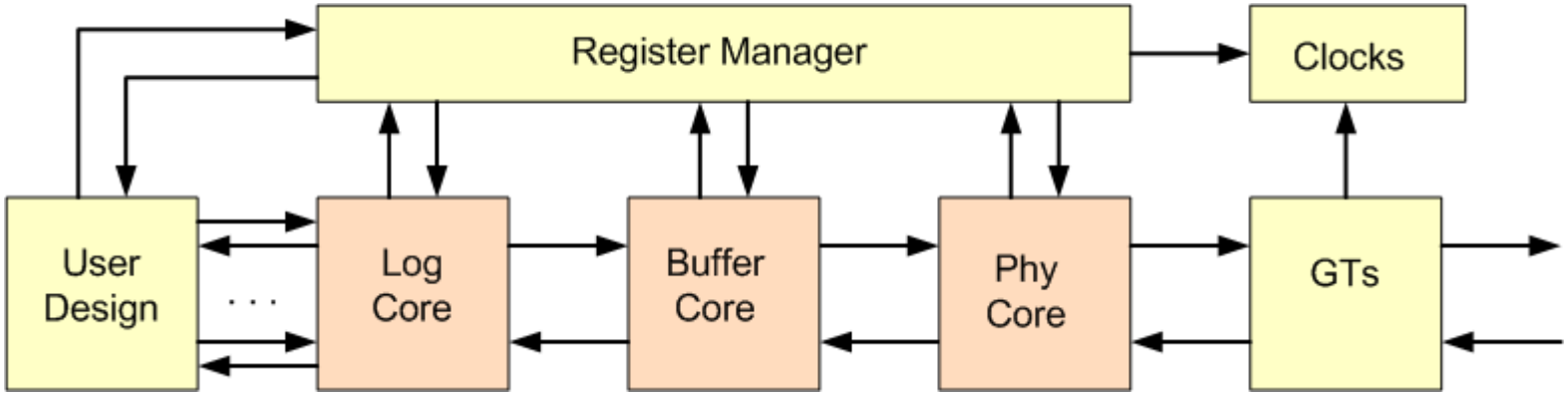
- Coverpoint bins are set when the covergroup is created

```
covergroup cg_fifo_count (int fifo_depth);
  coverpoint (full_count) {
    bins empty  = {0};
    bins in_use = {[1:fifo_depth-2]};
    bins full   = {fifo_depth-1};
  }
endgroup
```

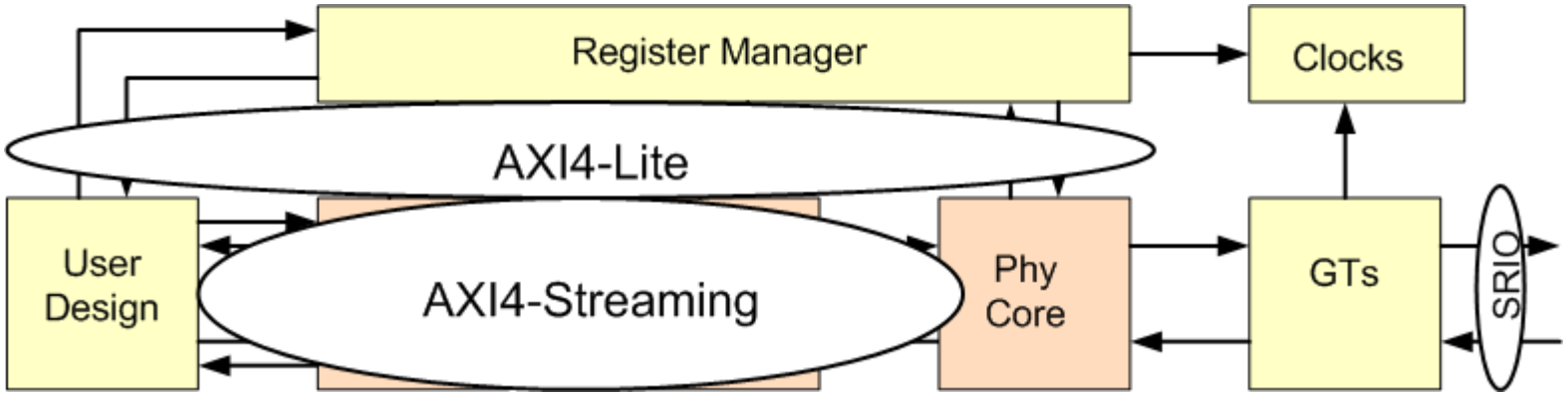
- Generate conditionals result in dead code

```
generate if (MODE1) begin
  // Active when Mode1 is 1, dead code when MODE1 is 0
end else begin
  // Active when Mode1 is 0, dead code when MODE1 is 1
end
```

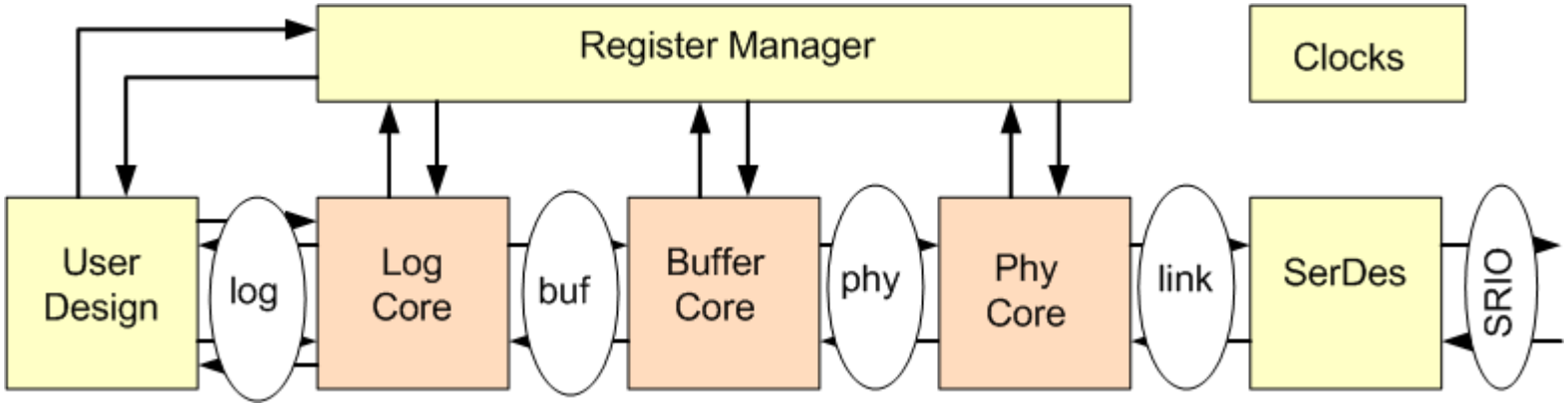
Serial RapidIO Core Architecture



Serial RapidIO Core Architecture



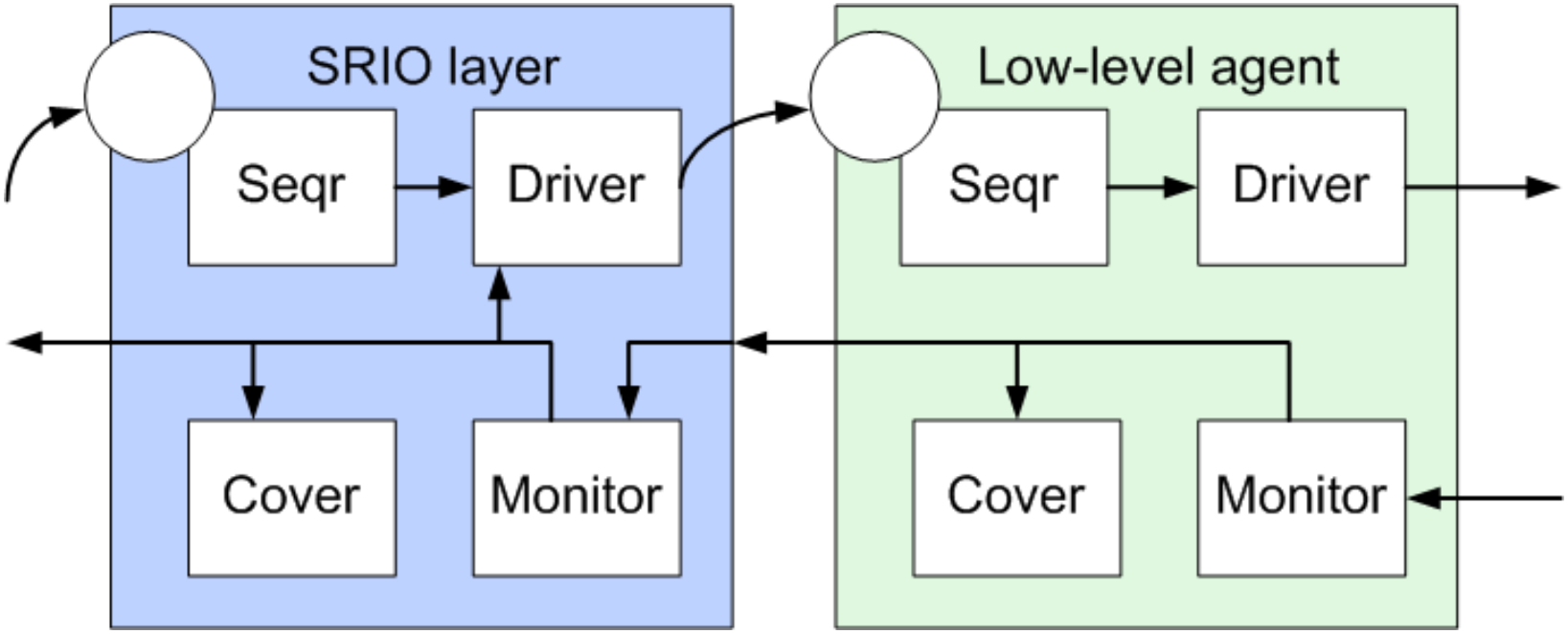
Serial RapidIO Core Architecture



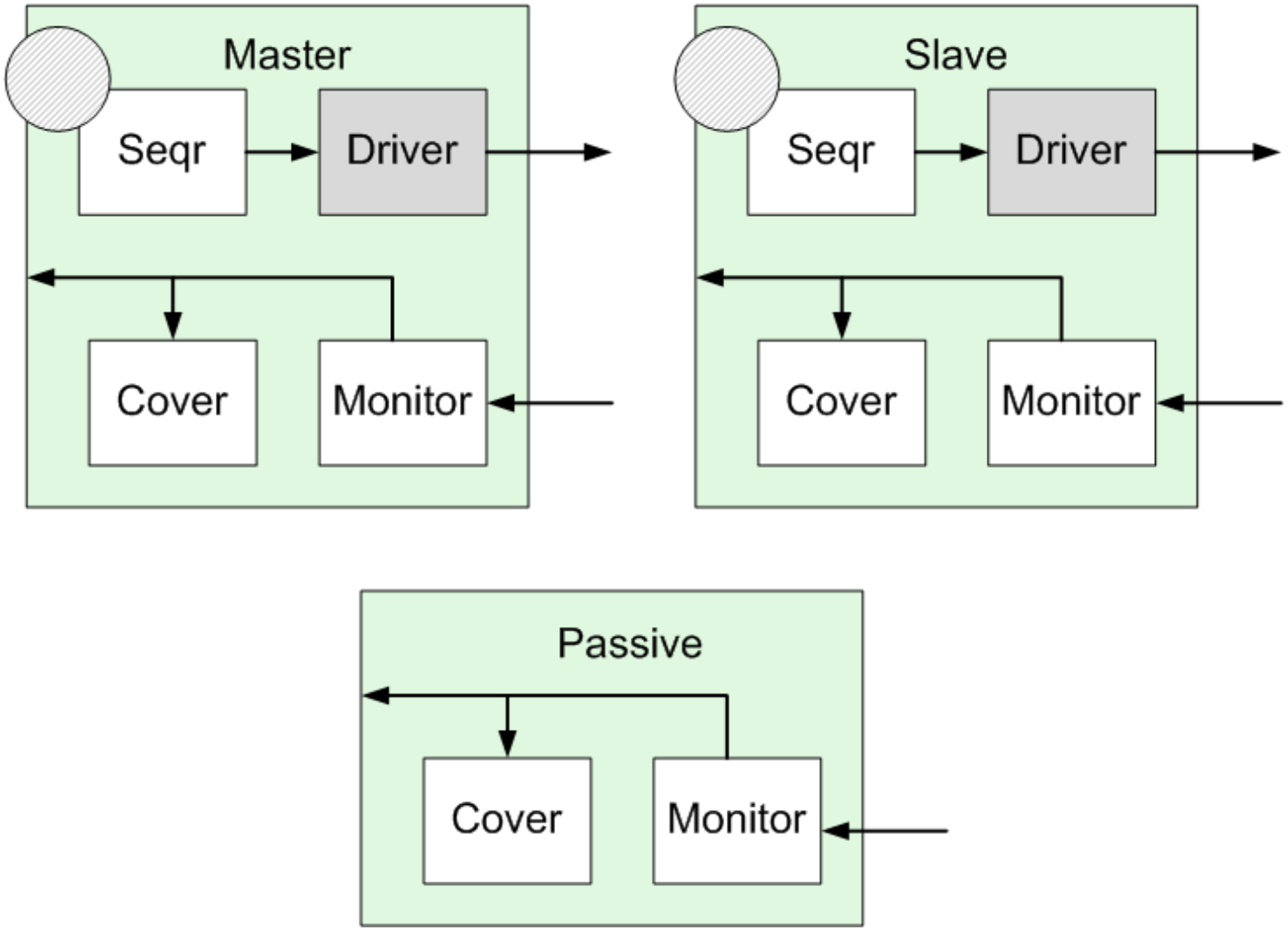
Serial RapidIO on top of AXI4-Stream

- 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

Reuse: layering



Reuse: mode


Master
Seqr
Driver
Cover
Monitor
Slave
Seqr
Driver
Cover
Monitor
Passive
Cover
Monitor