Tackling Register Aliasing Verification Challenges in Complex ASIC Design

Shan Yan, Jie Wu, Jing Li

Broadcom Limited

Agenda

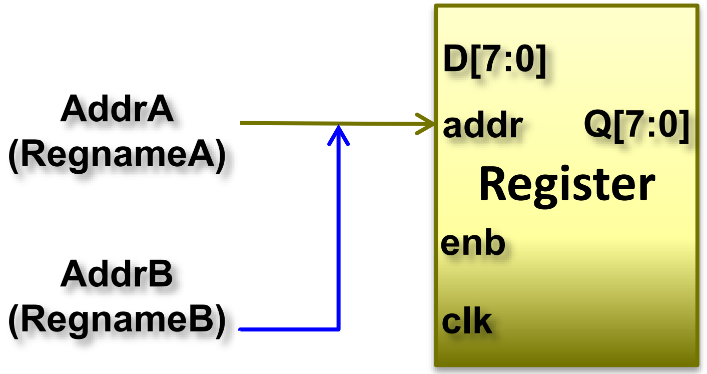
- Problem Description
- Motivation and Related Work
- Methodology Details
- Results
- Summary

Problem Description

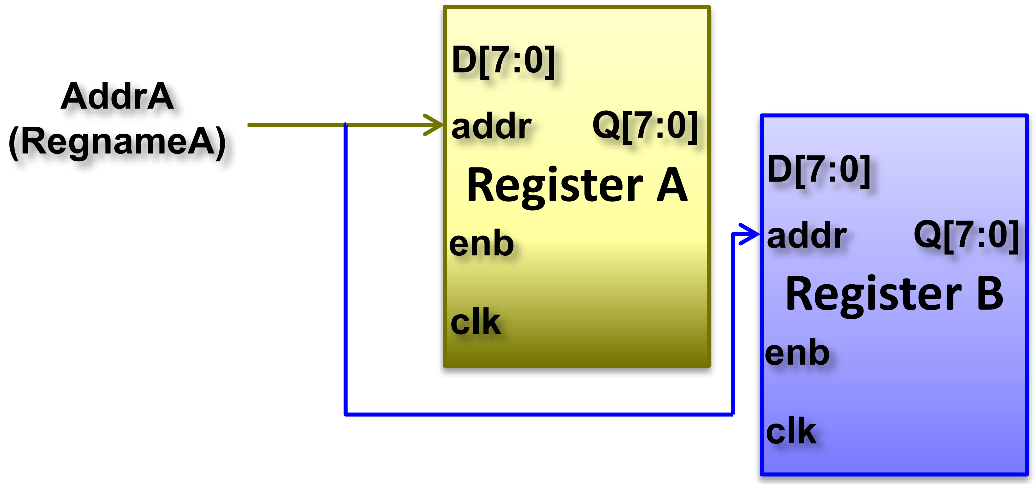
- Register design and implementation is one of the most important parts of today's hardware and IC designs.
- Registers contain the configuration setting of the hardware and are the basis of the hardware and software interface.
- Register verification is a significant part of the design verification problem. It is one of the first aspects of the design that must be tested because the rest of the semiconductor functionality depends on the accuracy of the register implementation.

Register Aliasing Issues

- A register location in the hardware design can be accessed through different symbolic names.
- One symbolic name and address is associated with multiple register locations in the hardware design.
- It can either be a hardware design choice or a hardware failure.



(a) Two addresses reach to the same register



(b) One address reaches to two registers

Register Aliasing Verification Challenges

- Today’s complex ASIC designs contain hundreds or thousands of registers.
- The traditional simulation-based verification method:
  - Usually based on the methodology of writing a unique value to one register and reading back the values from all registers and then comparing.
  - Very time consuming: O(n^2)
  - Hard to reach coverage completeness.

Verification Methodology

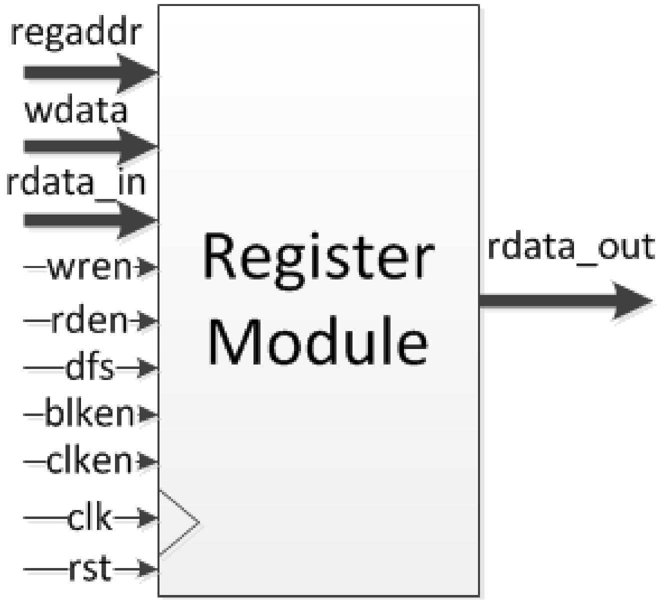
- Combines assertion-based technique and simulation to find register aliasing issues.
- Two parts:
  - Assertion models are developed to check read_enable and write_enable signals for the register access to make sure no more than one register access at any time.
  - Three register sequences are carefully designed to test the register read and write accesses.
- Whole procedure is automated.
- Performance improvement: O(n)

Register Access Modeling (1)

- Bus architectures are used in our designs.
- All registers are attached to the same bus for access.
- At any given time, only one register is accessed by the master on the bus.
- Write_enable and read_enable signals are checked for register accesses.
- An overall write_enable counter and a read_enable counter are maintained at the chip level to count register access on all registers.

Register Access Modeling (2)

- Register read_enable and write_enable checking



```
module typeA_reg_bind (clk, …, wr_en, rd_en,…);
assign reg_wr = wr_en & regen;

always @(negedge clk) begin
  if (reg_wr) reg_wr_d = 1;
end

always @(posedge clk) begin
  if (reg_wr & ~reg_wr_d) begin
    $display ($time, "%m : reg_wr is asserted.");
    `TB_TOP.reg_wcnt = `TB_TOP.reg_wcnt + 1;
  end
end
endmodule
```

Register Access Modeling (3)

- An overall write_enable counter and a read_enable counter are maintained at the chip level to count register access on all registers.

```
bit [15:0] reg_wcnt, reg_rcnt;
bit reg_rd_chk_on, reg_wr_chk_on;

initial begin
  reg_wcnt = 0;
  reg_rcnt = 0;
  reg_wr_chk_on = 1'b0;
  reg_rd_chk_on = 1'b0;
end

always @(reg_wcnt) begin
  assert(reg_wcnt <= 1)
  else `uvm_error(get_type_name(),
  $psprintf("!ERROR! : ……
end
```

```
always @(posedge reg_wr_chk_on)
begin
  reg_wcnt = 0;
  reg_rcnt = 0;
end

always @(reg_rcnt) begin
……
end

always @(posedge reg_rd_chk_on)
……
end
```

Register Access Modeling (4)

- Bind register aliasing checking model to registers

```
bind typeA_reg_module typeA_reg_bind u_typeA_reg_bind_inst(.*);
bind typeB_reg_module typeB_reg_bind u_typeB_reg_bind_inst(.*);
bind typeC_reg_module typeC_reg_bind u_typeC_reg_bind_inst(.*);
……
```

Register Read/Write Tasks

- Modified register read/write tasks using RAL to control register access checking.

```
task write_register (intfType intf, string regname = "undef", bit[31:0] wdata,
regIntfType reg_if = APB);
  ......
  reg_inst = get_reg_by_name (intf, regname);
  bit ok = 1'b1;
  ok &= uvm_hdl_deposit("system.reg_wr_chk_on", 1'b1);

  reg_inst.set(wdata);
  reg_inst.write(status, wdata, , regif2name(reg_inst, reg_if));

  ok &= uvm_hdl_deposit("system.reg_wr_chk_on", 1'b0);
endtask
```

Register Read/Write Tasks

- Modified register read/write tasks using RAL to control register access checking.

```
task read_register (intfType intf, string regname = "undef", output bit [31:0]
rdata, input regIntfType reg_if = APB);
  ......
  bit ok = 1'b1;
  ok &= uvm_hdl_deposit("system.reg_rd_chk_on", 1'b1);

  reg_inst = get_reg_by_name (intf, regname);
  reg_inst.read(status, rdata,,regif2name(reg_inst, reg_if));
  reg_inst.set(rdata);

  ok &= uvm_hdl_deposit("system.reg_rd_chk_on", 1'b0);
endtask
```

Register Access Sequences

- All sequences use the UVM Register Access layer (RAL) model.
- Combine both frontdoor and backdoor access.
- Three sequences are created to work with our register access checking models.
  – Seq 1: backdoor write and frontdoor read sequence
  – Seq 2: frontdoor write and backdoor read sequence
  – Seq 3: frontdoor write and frontdoor read sequence

Results – Setup (1)

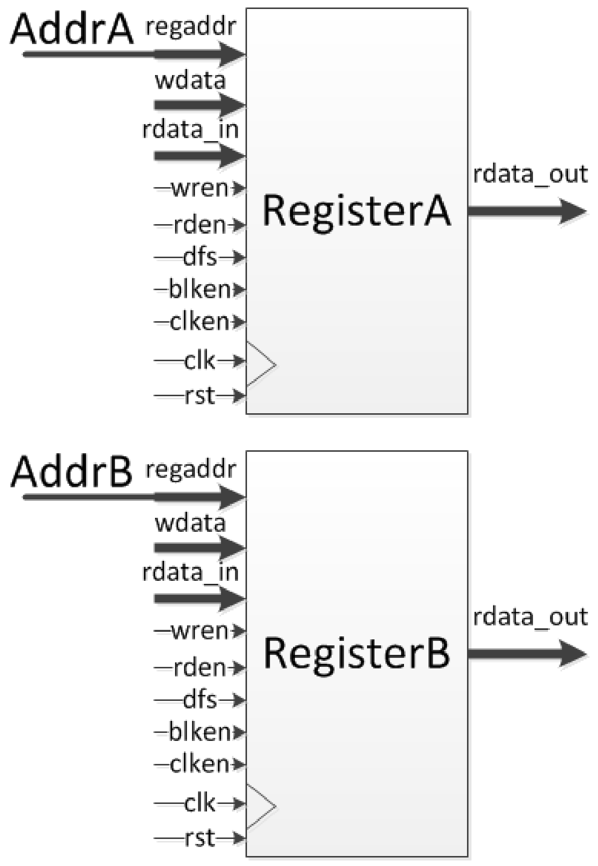
- In our designs, an internal format – RDB (Register Database Format) is used to define registers.
- Registers in the design are automatically generated from RDBs.
- The whole register verification procedure is automated. Scripts are developed to:
  - Generate UVM RAL model.
  - Extract registers' RTL hierarchical paths in the design for backdoor access.
  - Generate sequences and tests.

Results – Setup (2)

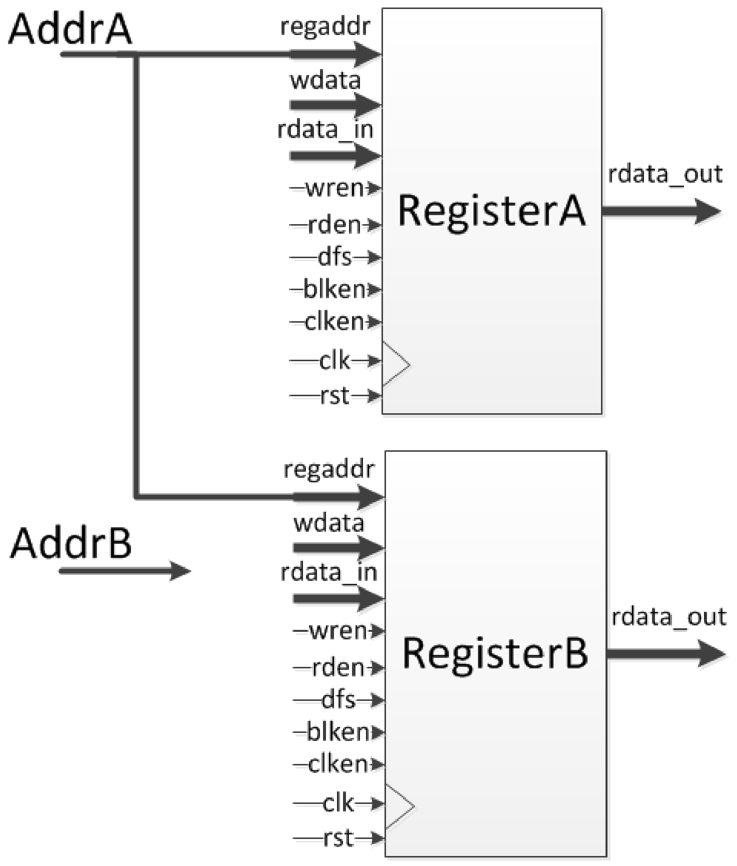
- The methodology is applied to three projects of our high-speed interconnect physical layer products.
- Each project contains thousands of registers.
- Several register aliasing bugs were found in the earlier stage of the design.
- Performance is largely improved. Saved us a lot of verification time.

Address Decoding Error

AddrA can access both registers but AddrB can access none.



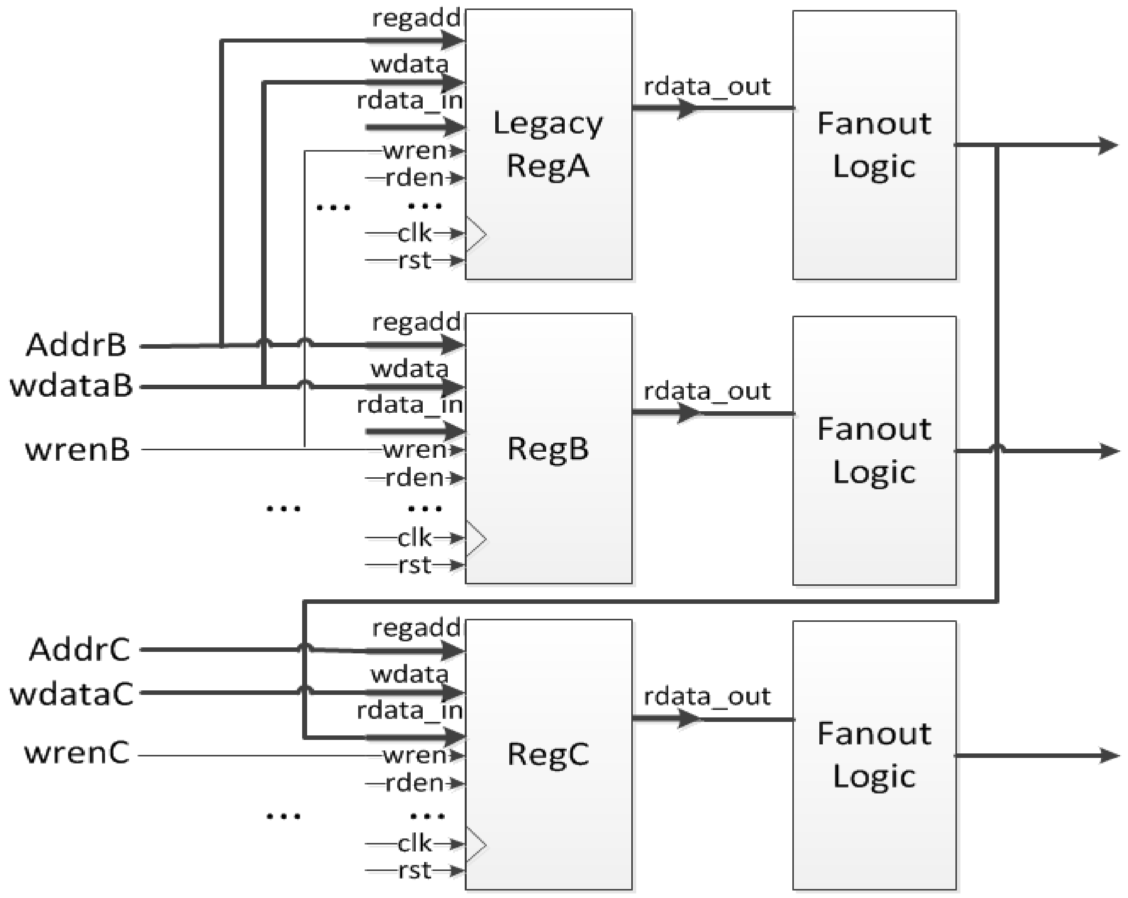
(a) Design Specification



(b) Design implementation

Duplicated Registers Bug

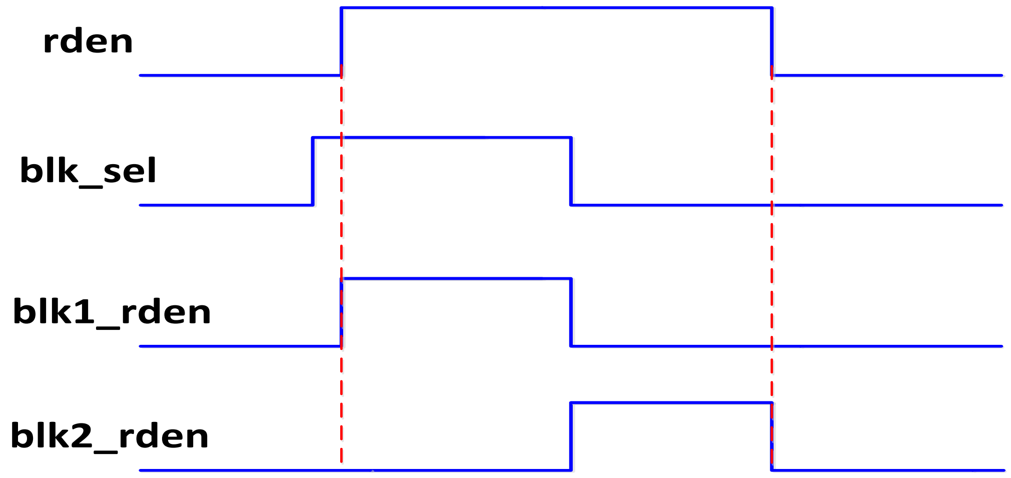
Legacy registers' addresses were used for new registers. But design logic was not removed, thus affect the new design logic.



Multiple Registers Read/Write Bug

Logic error of read_enable and write_enable signals resulted in two register accesses to two blocks.

```
reg rden, blk_sel;
wire blk1_rden, blk2_rden;
…
blk1_rden = rden & blk_sel;
blk2_rden = rden & ~blk_sel;
```



Performance Comparison

Projects	Average num. of registers in one block	Average runtime of traditional register aliasing test sequence (hours)	Average runtime of register enable signal checking sequence (hours)	Improvements ($Runtime_{traditional}/Runtime_{new}$)
Project A	126	53.55h	1.70h	31.5x
Project B	192	66.86h	1.45h	46.1x
Project C	244	108h	1.77h	61x

- **Compared our methodology with traditional reg aliasing check sequence.**
- **Kept reg numbers less than 300 per block for traditional sequence to work.**
- **For our methodology, regs can be grouped into very fine granularity blocks.**

Summary

- The register aliasing issue is very hard to capture with today's complex ASIC designs.
- The traditional simulation-based method is very time consuming and hard to achieve coverage completeness.
- A unique methodology combining assertion-based register read/write enable signal checking and simulation-based register access is presented.
- Bugs found in our projects were hard to find using the traditional method.
- 31.5x to 61x performance improvement were achieved.

Thank You!