FPGA-based Clock Domain Crossing Validation for Safety-Critical Designs

Alexander Gnusin
Agenda

• CDC Design Issues: an Overview
• Current CDC Verification Process
  – Static Verification
  – Dynamic Verification
  – Formal Verification
• CDC Verification applied to FPGA designs
  – FPGA-specific CDC verification: issues and solutions
  – CDC jitter emulation to enhance CDC influence
• Enhanced FPGA-specific CDC verification process
• Summary
CDC Design Issues: Metastability

- Signals crossing clock domain are asynchronous.
- Asynchronous signals cause Setup & Hold violation at capturing FF.
- As a result, capturing FF may oscillate for some time, producing intermediate logic values.
- Intermediate logic values, when propagating forward, may cause an “avalanche effect”, leading to functional failures and chip overheat.

![CDC Design Issues Diagram]
Metastability solution: Synchronizer

- Synchronizer design:
  - \( R_1 \) close to \( R_2 \), small wiring delay
  - Almost a receiving clock period given to oscillations to settle down
  - \( R_2 \) captured deterministic (settled down) value

- Prevents metastable values spread to fanout logic

- Still, does not guarantee functional determinism
Non-determinism with Synchronizers

• When signal changes at Sample & Hold capturing clock region: no guarantee that changed value captured at receiving flip-flop

• Introduces non-determinism in deterministic design functionality

• Functional CDC verification: CDC non-determinism not affecting deterministic design behavior

![Diagram](attachment:image.png)
Transferring related signals

• Synchronizer delay is unpredictable:
  – No issues for independent single-bit controls
  – May cause timing shift of functionally related signals

• Care should be taken to verify that non-deterministic timing shift not causing functional issues

• Example: related enables timing shift causes undesired glitch at receiving domain
Related Control Signals in CDC

- **Convergence**: signals converge in driving clock domain

- **Re-convergence**: signals converge in receiving clock domain

- **Divergence**: signal diverges to multiple synchronizers
Current CDC Verification Process

- Static Verification with LINT tools
- Dynamic Verification (Simulation) with CDC-related assertions & coverage & models
- Formal Model Checking with CDC assertions & models
- Lab Testing
Static Verification with LINT Tools

- Topology-based verification of constrained RTL design:
  - Extract clock domain crossings
  - Validate the synchronizer patterns correctness at clock domain crossings
  - Extract divergence, convergence and reconvergence cases
  - Generate assertions for functional verification
- Problems:
  - Lint Tools cannot fully verify CDC issues
  - Design constraints may be wrong & incomplete
Simulation-based CDC verification

• Add LINT-generation assertions to verification environment
• Optionally, use synchronizers with random CDC jitter modeling
• Ensure functional tests & regressions pass with no assertion failures
• Use functional coverage to improve tests quality for CDC checking

• Problems:
  – Need to verify all possible delay combinations on clock domain crossings
  – RTL checking may miss timing issues & CDC-specific test scenarios
  – Gate-level sims: long simulation times, small amount of tests
CDC Formal Analysis

- Add LINT-generation assertions to verification environment
- Optionally, use synchronizers with random CDC jitter modeling
- Add assumptions to constrain design behavior
- Problems:
  - May not prove all CDC assertions due to large design size
  - Formal constraints (assumptions) may be wrong & functionally incomplete
Lab Testing for CDC Verification

• Lab Testing verifies real design, including functionality and timing
• May reveal CDC-related issues right away
• However, some CDC-related issues may remain undiscovered due to:
  – Clock sources variations
  – Process, Voltage, Temperature variations during device usage
CDC Verification Completeness

• Each CDC verification method:
  – Not providing complete confidence
  – May be based on wrong & incomplete input data (design constraints, test vectors, assumptions)
• Mission-critical multi-clock FPGA designs require another “integrated” CDC verification method
• Solution: CDC Jitter emulation in FPGA testing
CDC Jitter Concept

• CDC jitter models unpredictable synchronizer delay when metastability occurs as a result of Sample & Hold Violation at first synchronizer register:

• CDC jitter can be modeled for simulation & formal analysis:
  – Minimum 3 FF stages in synchronizer
  – Using $random function
Synthesizable CDC Jitter model

- Use synthesizable Ring-Oscillator randomizer to randomly select R2 or R3 stages values
- Asynchronous Enable input to enable & disable randomization for debug purposes
One-bit Randomizer Design

• Use Ring oscillator for in-FPGA Randomization:
  – Non-deterministic value captured at R1 due to metastability effect
  – Single NAND in the ring to increase the frequency of metastability events
  – Two flip-flops (synchronizer) required to stop metastable values propagation
  – Asynchronous Enable signal to enable / disable oscillations:
    • Saves power
    • Sets randomizer to pre-defined value
Single-bit Randomizer Code

- Use “ASYNC_REG” in Xilinx and similar attributes in Intel FPGA to place synchronizer registers “as close as possible” (in same slice)
- Use attributes / LCELL instance to prevent ring oscillator optimization (removal)
- Use ALLOW_COMBINATORIAL_LOOPS attribute on ring oscillator net

```verilog
module bit_rng (input clk, en, output rnd_out);
    (* ASYNC_REG = "TRUE" *) reg meta, sync;

    // inverter chain oscillator
    (*DONT_TOUCH= "true"*) wire osc;
    assign osc = ~(osc & en);

    always @(posedge clk) begin
        meta <= osc;
        sync <= meta;
    end
    assign rnd_out = sync;
endmodule
```

Constraint

```
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets osc]
```
module rnd_sync(
    input clk, din, en, output dout);
(* ASYNC_REG = "TRUE" *) reg meta, sync1, sync2;
wire sel;
bit_rng sel_rng (clk, en, sel);
always @(posedge clk) begin
    meta  <= din;
    sync1 <= meta;
    sync2 <= sync1;
end
assign dout = (sel)? sync1 : sync2;
endmodule
Re-usable Synchronizer Model

- Re-use synchronizer model between Lab Verification, Simulation and Formal Analysis:
  - Lab Verification: Instantiate bit randomizer instance
  - Simulation: randomize MUX select signal
  - Formal Analysis: Allow Formal tools to treat disconnected MUX select signal as an unconstrained input signal

module rnd_sync(
  input clk, din, en, output dout
);
(* ASYNC_REG = "TRUE" *)
reg meta, sync1, sync2;
wire sel;

`ifdef SYN
bit_rng sel_rng (clk, en, sel);
`elseif FORMAL
  // nothing : sel becomes primary input for formal tools
`else
  // Simulation, random selection
  always @(posedge clk)
    sel <= ($random)%2;
`endif

always @(posedge clk) begin
  meta <= din;
  sync1 <= meta;
  sync2 <= sync1;
end
assign dout = (sel)? sync1 : sync2;
endmodule
Using Sync Models for Reconvergence

- Using LINT tools, identify reconvergence logic & related synchronizers
- For each reconvergence case, replace related synchronizers with synchronizer models
- For each reconvergence group, tie synchronizer models enable together and connect them to dedicated CSR
- Enable / Disable each reconvergence group for verification / debug purposes
Enhanced CDC Verification Flow

• Use LINT tools to verify CDC topologies
• Replace all synchronizers with synchronizer models
• Group and connect sync models enables to dedicated control regs
  – group sync models enables related to same reconvergence logic
• Implement FPGA with Sync models, turn on all sync enables and run Lab Testing
• In a case of failure, control sync enables values to localize the problem
• Use simulation & formal methods to identify CDC issues in failing design part
Enhanced CDC Verification Challenges

• In Lab Testing, hard to get synchronizer delay combinations causing design failure
  
• Possible solution:
  – Add Shift Register LUT to each synchronizer
  – Stop design clock on CDC assertion failure
  – Use ChipScope to read out SRL values

• Problems:
  – Need to introduce more complicated logic to design
  – CDC-related assertion failure may occur much later in design: SRL length may not be sufficient
Localizing CDC failures

- Use LINT tools to identify reconvergence logic
- Connect sync enables in same reconvergence group to same control
- For other CDC cases, group sync enables connections following design hierarchy
- Upon CDC failure:
  1. Disable all Sync randomizer to guarantee Failure belongs to CDC
  2. Gradually re-enable synchronizer groups to localize failure
Summary

• Synthesizable CDC Jitter models enable extensive CDC verification for mission-critical FPGA designs

• CDC Jitter models in implemented FPGA design to enhance CDC effect, verifying hidden corner cases in real hardware

• Complements and enhances current CDC Linting & Simulation & Formal verification process

• Applicable for mission-critical verification of both FPGA and ASIC designs, with FPGA prototyping
Thanks for your attention!
Any questions?