Eliminate Silicon Failures by 100% CDC Signoff Confidence Using Static-Aware Synthesis Followed by Netlist CDC Verification

Yujing Feng[1*], Donghui Xie[2], Qibin Chen[1], Yuan Wen[1], Yunfeng Shi[1]
[1]ByteDance Inc [2]Synopsys
{fengyujing.colleen,chenqibin,wenyuan.20210901,shiyunfeng.mark}
@bytedance.com, DongHui.Xie@synopsys.com

Abstract

When implementing an SoC chip, Clock Domain Crossing (CDC) verification guarantees no metastability-induced bugs on the clock domain crossing paths. Typically, we perform CDC checks at the RTL layer based on the understanding that synthesis tools will not change the functional behavior of RTL code. LEC tools further guarantee the functional equivalence between the RTL code and the netlist. This also helps in the shift-left of CDC verification.

However, as the designs and implementation flows become more complex, significant logic optimizations are introduced during RTL synthesis and backend flows. The primary objective of the implementation flow is to meet the PPA (power/performance/area) goals, and current implementation flows do not have the sense of the clock domain crossing information. Thus, current implementation flows might perform improper transformation and optimization on the CDC paths. Although these improper transformations and optimization do not change the functional behavior, a change in the location of CDC blocking logic, as well as implementing glitch-free mux with glitchy And-Or-Inverter structure, may lead to the introduction of new potential CDC issues and make the actual silicon unstable. Even when the RTL & Netlist LEC are performed successfully, new CDC issues can still be introduced and observed on the netlist.

In this paper, we first discuss the CDC issues during RTL to Netlist transformation during synthesis with detailed examples, such as glitches introduced on mux logic, glitches on implementation inferred clock gating logic. Further, we will present a proposed solution for exposing the potential silicon failures with netlist CDC Verification flow. Finally, the innovative static and synthesis cross-technology solution, 'static-aware synthesis flow' will be discussed.

1. Introduction

1.1 Background

With the development of technology, the integration level of digital circuits is increasing, and the design is becoming more complex. In a typical SOC chip, there are often multiple clocks, which can be either synchronous or asynchronous to each other. Clock domain crossing (CDC) issues may occur if logic paths exist between asynchronous clocks.

CDC check is the verification of asynchronous paths in a chip. Since asynchronous clocks do not have a fixed phase relationship, it will likely result in setup/hold violations, leading to metastability issues. We adopt synchronous design methods to ensure that metastability does not propagate randomly in the circuit, thus avoiding functional issues.

Currently, most digital designs are based on RTL design. It is difficult to achieve 100% "clock domain crossing" coverage through direct RTL checks, especially in modern SOC designs that integrate many IPs. The verification of synchronous circuit design is performed using dedicated EDA tools for CDC checks. This article will illustrate the process using Synopsys's VC SpyGlass CDC tool.

The primary function of VC SpyGlass CDC is that it has predefined rules that check the design to see for compliance. If there are any violations of the rules, it will report these violations to the designers for debugging. Different rules are divided into several major categories, referred to as goals. It will first perform setup rule checks on the design and constraints, such as whether each flop has a defined clock, whether the reset signal is determined, etc. After passing the setup rule checks, it will perform a series of CDC checks if signals cross clock domains without going through synchronizers, if the same signal is synchronized multiple times within the same clock domain, etc. All designs that do not comply with the rules will be reported, and they can be presented to designers in the form of schematic, spreadsheet, and text reports. There will also be detailed rule descriptions, explanations of why the design violates the rule, and corresponding modification suggestions. VC SpyGlass CDC leverages inbuilt formal methodology under the hood leading to reduced dependency on writing test cases and running simulations to find CDC issues. Using VC SpyGlass CDC can help identify CDC issues earlier, more comprehensively, and faster.

1.2 Motivation

As mentioned earlier, regardless of the EDA company, the current industry-standard CDC process involves performing checks on the RTL. Over the years, we have found that the main problem with this RTL checking flow is that during synthesis, RTL to the gate-level circuit conversion potentially leads to unpredictable behavior for CDC paths already proven during RTL CDC verification. These unexpected CDC paths cannot be detected during RTL CDC check phase because these do not exist at RTL. We have mainly identified the following two types of issues:

(1) The first issue is the glitch propagation across clock domains caused by the MUX logic in the asynchronous FIFOs. As shown in Fig.1, the glitch issue lies in the MUX logic between the green clock domain and the red clock domain.

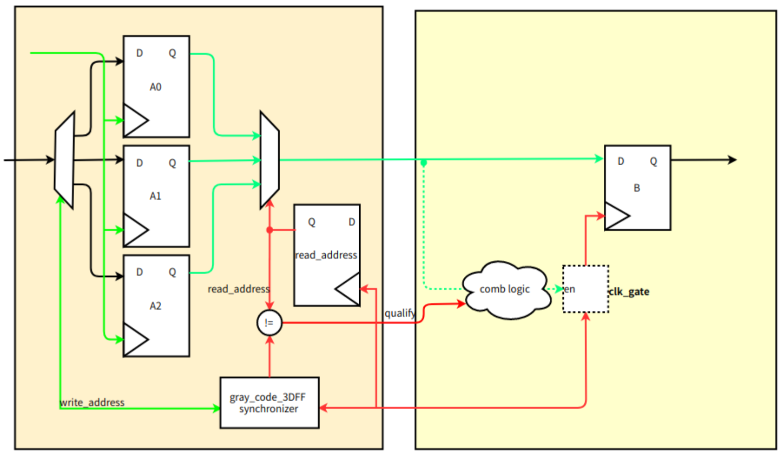


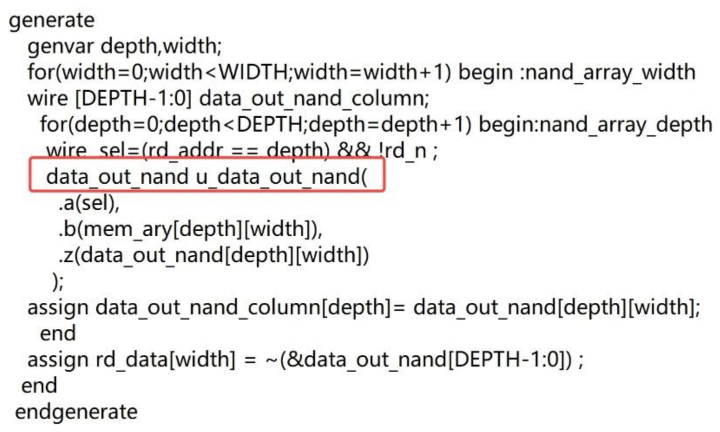
Fig.1 A circuit diagram of an asynchronous FIFO.

The original RTL code is expressed as Fig.2 for this clock domain crossing MUX logic.

```
generate
genvar depth,width;
 for(width=0;width<Width;width=width+1)
    begin :nand_mux_stage_2
      wire [DEPTH-1:0] data_out_nand;
    for(depth=0;depth<DEPTH;depth=depth+1)
      begin:nand_mux_stage_1
        wire sel =(rd_addr == depth);
        assign data_out_nand[depth]= ~(sel & mem_ary[depth][width]);
      end
     assign rd_data[width] = ~(&data_out_nand[DEPTH-1:0]) ;
 end
endgenerate
```

Fig.2 Verilog code for mux logic in the asynchronous FIFO.

The corresponding synthesized design for the above RTL code comprises a series of logic gates involving MUX logic. This logic cannot guarantee that the final selection stage has a qualifier on each bit, which may result in glitches propagating to the downstream clock domain when the select signal remains unchanged while register A0~A2 flip. If the destination clock samples this glitch, it will cause the design to fail. These risky paths due to glitches can be detected using the netlist CDC tools. As shown in Fig.3, the corresponding modification approach is to instantiate the "AND" logic between sel and data using standard library cells, ensuring that this layer of qualifier combination logic isn't be optimized out.



```
generate
  genvar depth,width;
  for(width=0;width<WIDTH;width=width+1) begin :nand_array_width
  wire [DEPTH-1:0] data_out_nand_column;
    for(depth=0;depth<DEPTH;depth=depth+1) begin:nand_array_depth
     wire  sel=(rd_addr == depth) && !rd_n ;
     data_out_nand u_data_out_nand(
       .a(sel),
       .b(mem_ary[depth][width]),
       .z(data_out_nand[depth][width])
      );
  assign data_out_nand_column[depth]= data_out_nand[depth][width];
    end
  assign rd_data[width] = ~(&data_out_nand[DEPTH-1:0]) ;
 end
endgenerate
```

Fig.3 The proposed modification method for the MUX glitch issues.

(2) The second issue concerns the clock enable logic inserted by the synthesis tools in front of the destination clock domain.

As shown in Fig.4, it is a typical structure of an asynchronous FIFO, and the signals passing from the source (green) clock domain to the destination (red) clock domain is qualified. However, the synthesis tool automatically inserts a clock gate cell to the destination register B during the optimization phase. In the synthesized design, one possible scenario is that the enable input of this clock gate cell is driven by the combinational logic output from the asynchronous source clock domain. In this case, the timing violation at the enable input of the clock gate cell may cause glitches at the clock output, which poses a significant risk on the clock path and cannot be ignored.

This issue cannot be detected in the RTL CDC flow because the clock gate cell in the dashed box in the diagram is inserted during the synthesis process. Since the synthesis tool cannot obtain the masking information of the qualifier signals on the asynchronous path when inserting the clock gate cells, the cross-asynchronous paths are split. As a result, the cross-asynchronous combinational logic with qualifier goes through the D input of the B register, while the cross-asynchronous combinational logic without a qualifier goes through the E input of the clock gate cell in front of the B register, causing the glitch hazard on the clock path.

This issue can be mitigated by following specific coding style guidelines. The following rules should be taken into consideration when coding the asynchronous logic:

- Avoid using cross-asynchronous combinational logic. Using qualify signals is preferable to synchronize asynchronous signals to the target clock domain. This can help avoid the synthesis tool splitting cross-asynchronous combinational logic when inserting the clock gate cell to optimize the clock gating ratio.
- Avoid using asynchronous signals as the conditional statements in the “else if” within an always block.

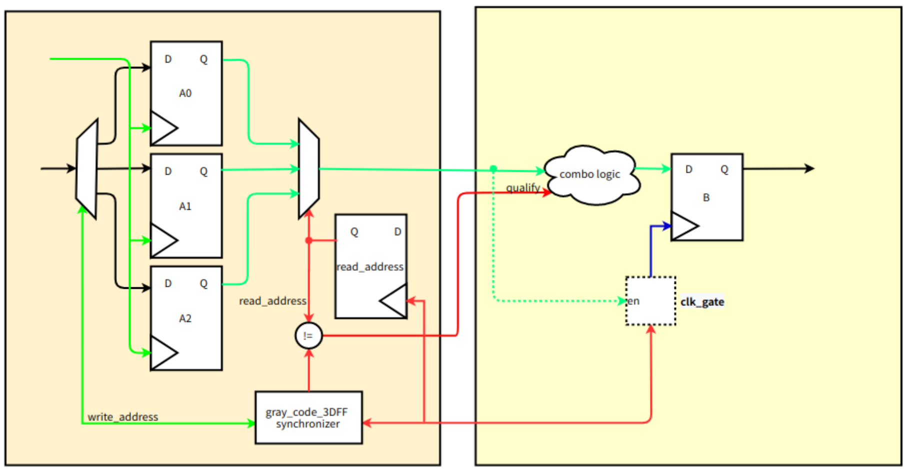


Fig.4 A circuit diagram of an asynchronous FIFO.

Both logic types are located at the transition points on the cross-clock domain path. Improper synthesized circuits may result in the propagation of glitches being sampled by the destination clock domain, leading to the metastable states in the logic of the destination clock stages. Checking CDC on the netlist can effectively identify the risks introduced by the two types of logic generated by the synthesis tool. Furthermore, for the second type of issue, a CDC-aware synthesis flow can be employed to avoid clock gating issues. Therefore, the following sections of this article will discuss specific issues identified during the execution of real projects, as well as provide detailed explanations on how to use netlist CDC flow to check and fix the corresponding problems.

2. Solutions with CDC Netlist Flow

2.1 Analysis of the Netlist CDC Flow

When there is a large amount of combinational logic between the source register and destination registers in a CDC transfer, the design behavior may differ from the expected behavior after the synthesis optimization. To avoid such inconsistencies, performing a CDC check on the generated netlist is necessary.

2.1.1 Functional Glitch Check Analysis

As shown in the previous Fig.1, it is a standard structure of an asynchronous FIFO circuit, where the source (green) clock and the destination (red) clock are asynchronous. When the data is stored in the green clock domain, the corresponding write address is converted to gray code and synchronized to the red clock domain. Each time the data is accessed from the FIFO, the relationship between the write address and the read address is used to determine whether the data in the FIFO can be accessed. If the access requirement of the FIFO is met, the data stored in the green clock domain FIFO will be selected and output to the destination red clock domain through combinational logic. From the perspective of the designer, the expected functionality of this cross-clock domain combinational logic is that when the read address remains unchanged, the combinational logic selects one data from one entry in the FIFO for output, which means that the read address and the FIFO entry should satisfy one-one mapping requirement. Otherwise, if the read address remains unchanged but the data flips, the red clock domain may receive data with glitches.

Fig.5 shows the actual circuit structure of the asynchronous FIFO after being optimized by the synthesis tool. There is a complex combinational logic between the source clock register and the destination clock register. If we rely on the manual inspection of the glitch hazard, it will take a significant amount of time to determine whether this circuit segment poses a risk of generating glitches. To quickly check if the cross-asynchronous domain logic circuits can produce unexpected glitches, the functional glitch check feature of the VC SpyGlass tool can be used.

Fig.6 shows the result of checking the netlist CDC without the functional glitch check feature, where the tool identifies paths in the netlist that may contain glitch circuits, most of which are related to asynchronous FIFO paths.

Fig.7 shows the paths where the tool identifies potential glitch circuits by using the functional glitch check feature. The number of paths indicated in Fig.6 and Fig.7 indicates that the formal check helps filter out a significant number of low-risk paths, reducing the effort of the manual check by designers.

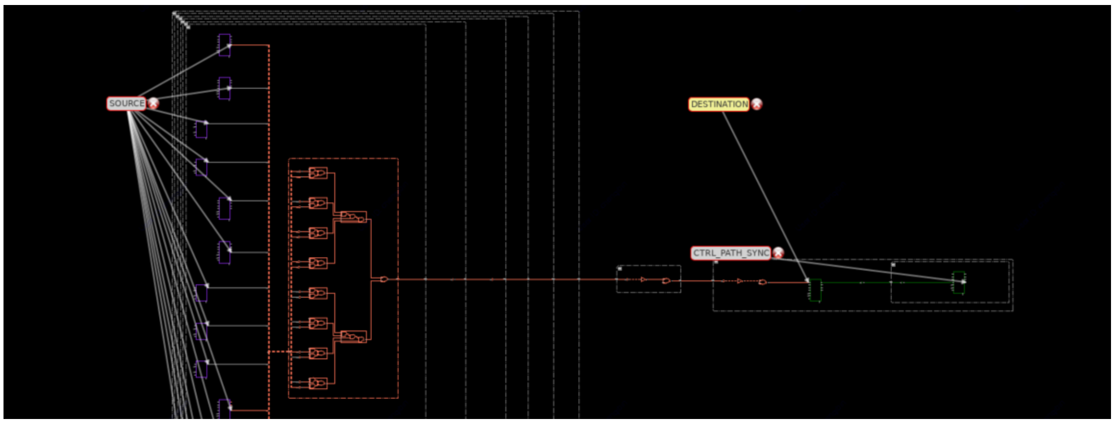


Fig.5 Synthesized Async FIFO circuit.

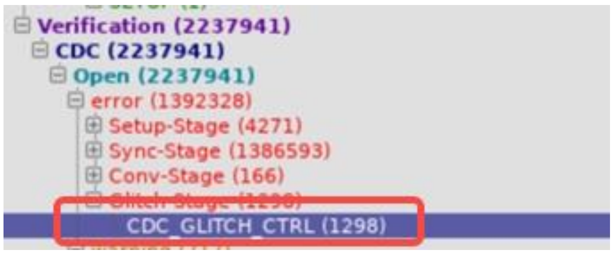
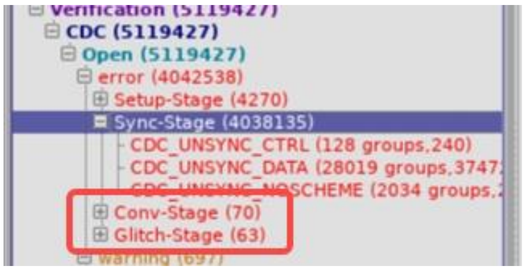

```
⊟ Verification (2237941)
  ⊟ CDC (2237941)
    ⊟ Open (2237941)
      ⊟ error (1392328)
        ⊞ Setup-Stage (4271)
        ⊞ Sync-Stage (1386593)
        ⊞ Conv-Stage (166)
        ⊟ Glitch-Stage (1298)
             CDC_GLITCH_CTRL (1298)
```

Fig.6 Netlist CDC result without functional glitch check.



```
Verification (5119427)
  CDC (5119427)
    Open (5119427)
      error (4042538)
        Setup-Stage (4270)
        Sync-Stage (4038135)
          CDC_UNSYNC_CTRL (128 groups,240)
          CDC_UNSYNC_DATA (28019 groups,3747
          CDC_UNSYNC_NOSCHEME (2034 groups,2
        Conv-Stage (70)
        Glitch-Stage (63)
      warning (697)
```

Fig.7 Netlist CDC results with functional.

2.1.2 Static-Aware Synthesis Flow

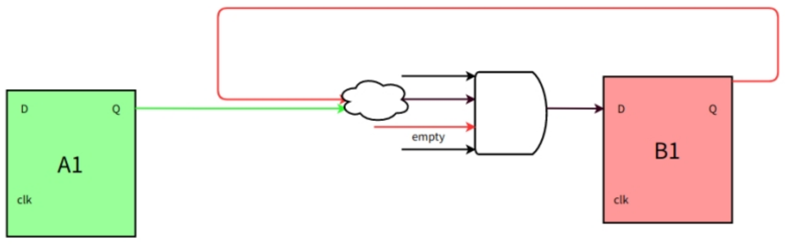


Fig.8 Data being qualified across the clock domain.

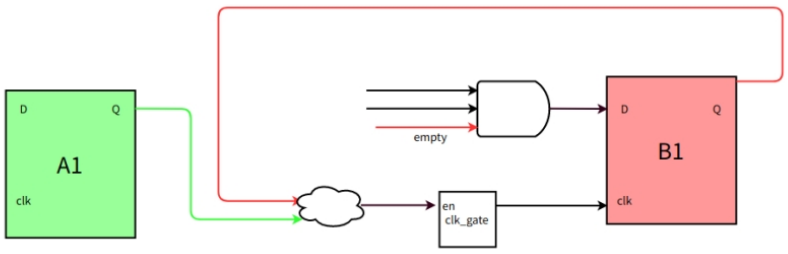
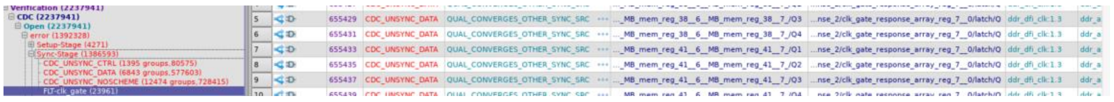


Fig.9 Synthesized circuit for the data being qualified across clock domain.

Fig.8 shows the structure when asynchronous data is synchronized to the asynchronous clock domain though the qualify signal. A register and B register correspond to asynchronous clock domains, and the empty signal in the B clock domain serves as the qualify signal for the output of the A register. When the output of A register changes, the empty signal is set to 0, blocking the asynchronous data transfer. When the output of A register stabilizes, the empty signal is set to 1, and the output of A register is synchronized to B clock domain. In the RTL state, this structure is just quite normal CDC logic.

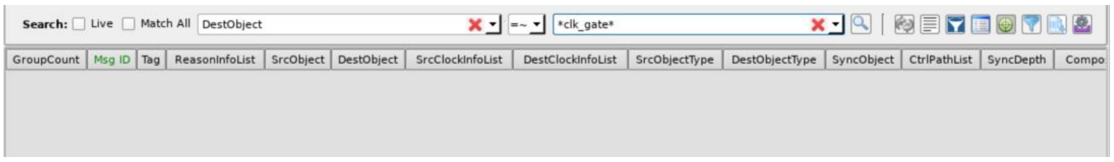
Fig.9 represents the circuit structure of Fig.8 after synthesis. After optimization by the tool, and clock gate cell is inserted between the A register and B register. This inserted clock gate cell controls the clock switch of the B register, which the qualified signal "empty" remains at the data pin of the B register. In this synthesized circuit, the empty signal cannot gate the changes of the A register, resulting in a timing violation at the clock gate cell, which may affect the functionality of the asynchronous FIFO.



```
Verification (2237941)
 CDC (2237941)
  Open (2237941)
   error (1392328)
    Setup-Stage (4271)
    Sync-Stage (1386593)
     CDC_UNSYNC_CTRL (1395 groups,80575)
     CDC_UNSYNC_DATA (6843 groups,577603)
     CDC_UNSYNC_NOSCHEME (12474 groups,728415)
      FLT-clk_gate (23961)

5  655429  CDC_UNSYNC_DATA  QUAL_CONVERGES_OTHER_SYNC_SRC  ...  ..._MB_mem_reg_38__6__MB_mem_reg_38__7_/Q3  ...nse_2/clk_gate_response_array_reg_7__0/latch/Q  ddr_dfi_clk:1.3  ddr_a
6  655431  CDC_UNSYNC_DATA  QUAL_CONVERGES_OTHER_SYNC_SRC  ...  ..._MB_mem_reg_38__6__MB_mem_reg_38__7_/Q4  ...nse_2/clk_gate_response_array_reg_7__0/latch/Q  ddr_dfi_clk:1.3  ddr_a
7  655433  CDC_UNSYNC_DATA  QUAL_CONVERGES_OTHER_SYNC_SRC  ...  ..._MB_mem_reg_41__6__MB_mem_reg_41__7_/Q1  ...nse_2/clk_gate_response_array_reg_7__0/latch/Q  ddr_dfi_clk:1.3  ddr_a
8  655435  CDC_UNSYNC_DATA  QUAL_CONVERGES_OTHER_SYNC_SRC  ...  ..._MB_mem_reg_41__6__MB_mem_reg_41__7_/Q2  ...nse_2/clk_gate_response_array_reg_7__0/latch/Q  ddr_dfi_clk:1.3  ddr_a
9  655437  CDC_UNSYNC_DATA  QUAL_CONVERGES_OTHER_SYNC_SRC  ...  ..._MB_mem_reg_41__6__MB_mem_reg_41__7_/Q3  ...nse_2/clk_gate_response_array_reg_7__0/latch/Q  ddr_dfi_clk:1.3  ddr_a
```

Fig.10 CDC report for the unqualified clock gating path.



```
Search: ☐ Live ☐ Match All  DestObject  =~  *clk_gate*

GroupCount | Msg ID | Tag | ReasonInfoList | SrcObject | DestObject | SrcClockInfoList | DestClockInfoList | SrcObjectType | DestObjectType | SyncObject | CtrlPathList | SyncDepth | Compo
```

Fig.11 CDC reports for the unqualified clock gating path after applying static-aware DC methodology.

Fig.10 shows the paths of asynchronous signals at the clock gate cell that are not gated by the qualified signals when checking the netlist with the VC SpyGlass tool. Fig.11 shows the results after using the static-aware DC flow and checking with the VC SpyGlass tool. By identifying specific paths and not inserting clock gating cells on such paths, the separation of the asynchronous signals and the qualified signal is avoided.

Without using the static-aware synthesis flow, the insertion ratio of the clock gate cell in this example block is 88.36%. After using the static-aware DC flow, the insertion ratio of the clock gate cell in this same example block is reduced to 86.29%. This reduction signifies the reduction in the number of potentially problematic CDC paths which might have led to silicon failure if not addressed.

2.2 Netlist CDC Issues

Modern implementation flows can involve significant structural changes via advanced techniques like logic optimization, clock gating insertion, retiming, ECOs(Engineering Change Order) insertion, Low Power cell insertion. As Figure 12 demonstrated, these structure changes account for a significant design percentage of which the RTL CDC signoff flow no longer can be relied upon, **Glitch and Clock Gating issues** mentioned in section 1.2 both belong to these typical kinds of CDC issues that pop-up on netlist. VC SpyGlass Netlist CDC solution provides complete 100% signoff on netlist CDC signoff and ensures such issues on netlist design won't occur.

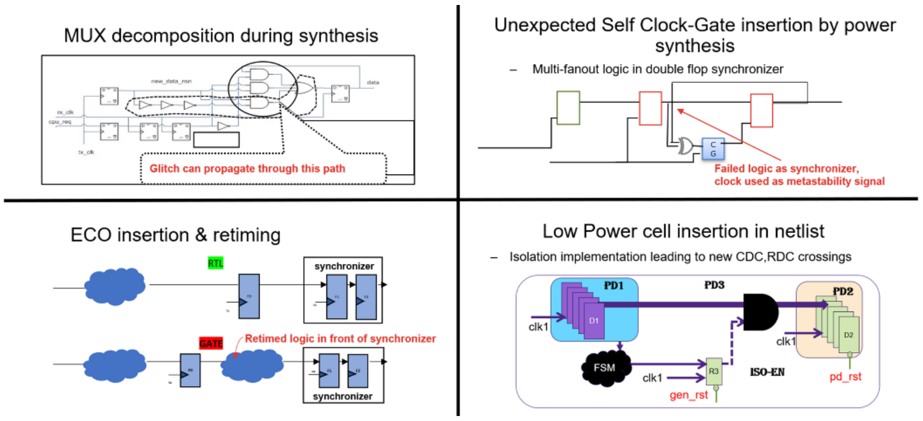

MUX decomposition during synthesis
new_data_nsn
data
rx_clk
cpu_req
tx_clk
Glitch can propagate through this path
Unexpected Self Clock-Gate insertion by power synthesis
– Multi-fanout logic in double flop synchronizer
C G
Failed logic as synchronizer, clock used as metastability signal
ECO insertion & retiming
RTL
synchronizer
GATE
Retimed logic in front of synchronizer
synchronizer
Low Power cell insertion in netlist
– Isolation implementation leading to new CDC,RDC crossings
PD1
PD3
PD2
clk1
D1
D2
clk1
FSM
clk1
R3
ISO-EN
gen_rst
pd_rst

Fig.12 Typical Netlist CDC Issues.

2.2.1 VC SpyGlass Netlist CDC Flow

As Figure 13 demonstrated, VC SpyGlass Netlist CDC Flow leveraging inbuilt PrimeTime netlist parser, significantly improving the design read runtime. It also supports Primetime setup reuse, which makes the setup flow of Netlist CDC much easier and more effective.

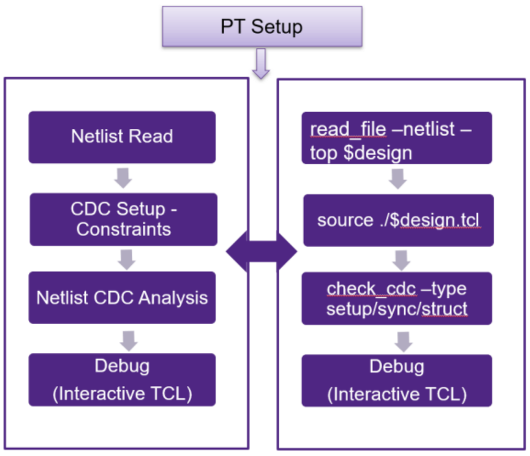


Fig.13 VC SpyGlass Netlist CDC Flow Diagram.

2.2.2 Functional Glitch Flow

During the Netlist CDC Signoff review stage, it is observed that the structure of CDC Glitch on netlist could become quite complicated. The typical case is that complex combinational logic exists on the glitch paths, making the glitch result review more tedious and time-consuming. This leads to a need for netlist CDC functional glitch analysis. As shown in Fig. 14, this flow filters the functional glitch-free CDC paths by leveraging Formal techniques, only indicating the real glitchy functional paths for the users to review. This ensures that the overall review cycle that the designer needs to spend is significantly reduced.

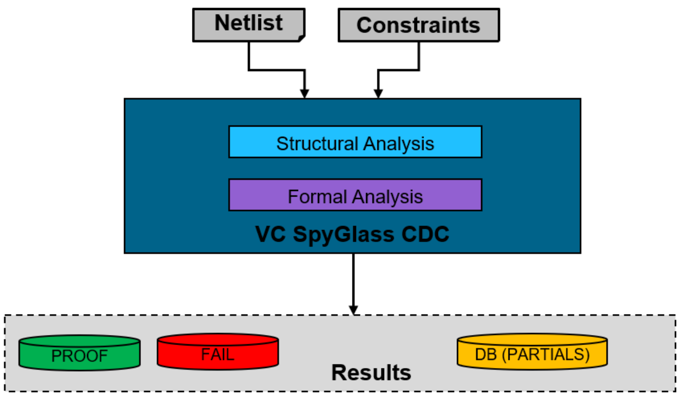


Fig.14 VC SpyGlass Netlist CDC-Functional Glitch Flow.

If the CDC glitch path is flagged as a formal pass, the glitch is just a glitchy structural path, and no real functional glitch will happen. Suppose the CDC glitch path is flagged as a formal fail, then VC SpyGlass CDC will generate the functional scenario causing the failure, and the scenario can be reviewed through Verdi waveform and schematic GUI. This is how VC SpyGlass Netlist CDC flow results in lower noise and makes the CDC signoff flow more effective and productive for designers.

2.2.3 Static Aware Implementation Flow

The clock gating issue mentioned in section 2.1.2 can be extracted as Fig. 15.

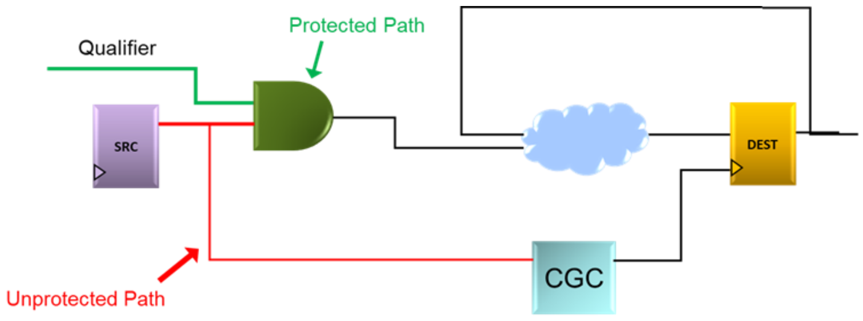


Fig.15 Asynchronous signal moves to Clock Gating Enable.

The synthesis tool is unaware of clock domain crossings, and optimizations are built to enhance Power, Performance, and Area (PPA) only. If clock domain crossing paths are restructured during these optimizations, it might cause asynchronous signal moving from Data Path to the Clock Gating Enable path. The RTL and Netlist still can pass the logic equivalence check, but Clock Gating Cell might go into metastable, which leads to gate-level re-verification of the netlist, it is time-consuming and might lead to re-synthesis if errors are found.
For this issue, the synthesis tool usually supports user constraint to exclude some specific logic applied on clock gating enable, that can stop the tool from inferring async clock gating logic as a clock enable logic. designers may proactively apply these types of constraints during synthesis, but the necessary effort can become significant if all these constraints need to be developed manually.

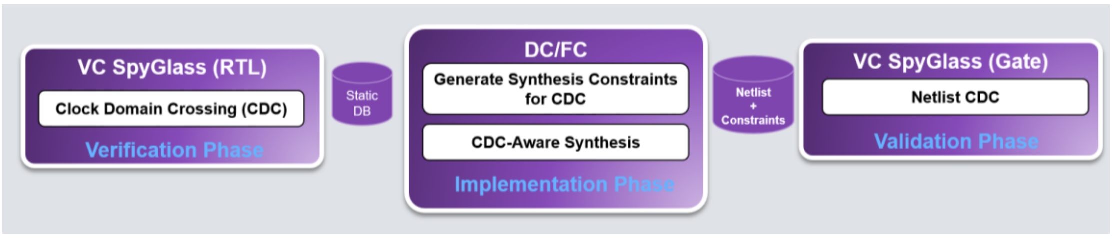


Fig.16 Static Aware Implementation Flow.

To address this challenge, Synopsys developed this static-aware synthesis flow (Fig. 16); VC SpyGlass analyzes the CDC paths from the static timing analysis perspective and provides an encrypted database to the synthesis tools. These constraints are then applied to the synthesis tools like Design Compiler NXT or the Fusion Compiler to avoid undesired optimizations/corruptions of the already pre-verified CDC paths. Hence, within this static-aware synthesis flow, the VC SpyGlass generated Static DB is used as input for an optimized synthesis flow that doesn't disrupt the pre-verified CDC paths. Designers can also run VC SpyGlass netlist CDC signoff flow reusing the

Static-Aware synthesis setup, to double confirm no CDC issue introduced by ECO/UPF logic insertion left on final netlist.

3. Conclusion

The functional glitch check technology of VC SpyGlass, leveraging inbuilt formal techniques, provides an accurate way to suppress the functional glitch-free paths on netlist designs. With this clock-domain crossing solution, designers can more efficiently locate the netlist CDC path where the real glitch problem exists. For risky paths, designers can optimize both the design and implementation to eliminate Netlist CDC Glitch Risk on Silicon.
Similarly, Static Aware Implementation Flow leverages proven VC SpyGlass CDC signoff technology in Design Compiler/Fusion Compiler implementation stage; this cross-technology solution obtains the clock domain information through VC SpyGlass CDC engine, predictably controls the gated clock optimization, creatively provides the user with the safety RTL to Netlist conversion.
These cross-technology solutions innovatively help designers eliminate silicon failures introduced by the transition from RTL to Netlist and lay a solid foundation for silicon timely tape-out and mass production.

4. Acknowledgement

The authors would like to thank Synopsys VC SpyGlass Experts Paras Mal Jain, Deepak Ahuja, Sudeep Mondal, Anshu Malani for the solid technical support for Functional Glitch check; thank Kenneth Trejos, Srinidhi Parshi and Renu Mehra for Static Aware Implementation Flow support; also, sincere thanks to Kenny Kwan, Xuesong Li from Synopsys SRG team for all the discussion and cooperation during Static Aware Flow deployment. Finally, special thanks to Kangliang Wei and Jingwei Zhu from the ByteDance Implementation team for all the collaboration during the Static Aware Flow deployment.

5. References

[1] Synopsys. (2023) VC SpyGlass Clock Domain Crossing LCA Features Guide, version U-2023.03.

[2] Synopsys. (2023) Static Aware DC FC Flow.