Automatic Generation of Formal Properties for Logic Related to Clock Gating

Shuqing Zhao, Shan Yan

Broadcom Corporation

Agenda

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

Problem Description

- Mobile computing systems - Low power is important
- Reducing dynamic power - Clock gating is the solution
- Integrated Clock Gating (ICG) cells are inserted manually by the RTL designers in our mobile SoC chips.
- Two types of clock enable control are used:
  - Software controlled:

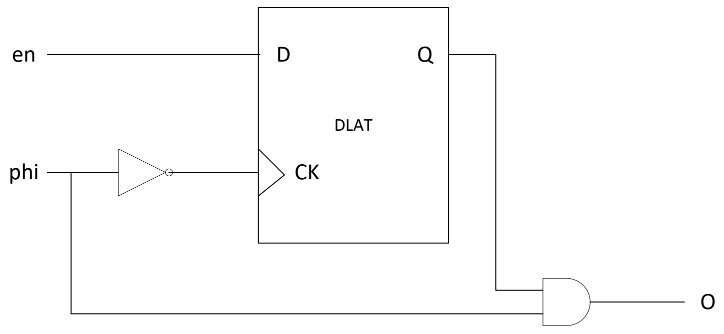
    use a programmable register bit

    to drive clock enable signal
  - Hardware controlled:

    use automatic clock gating logic

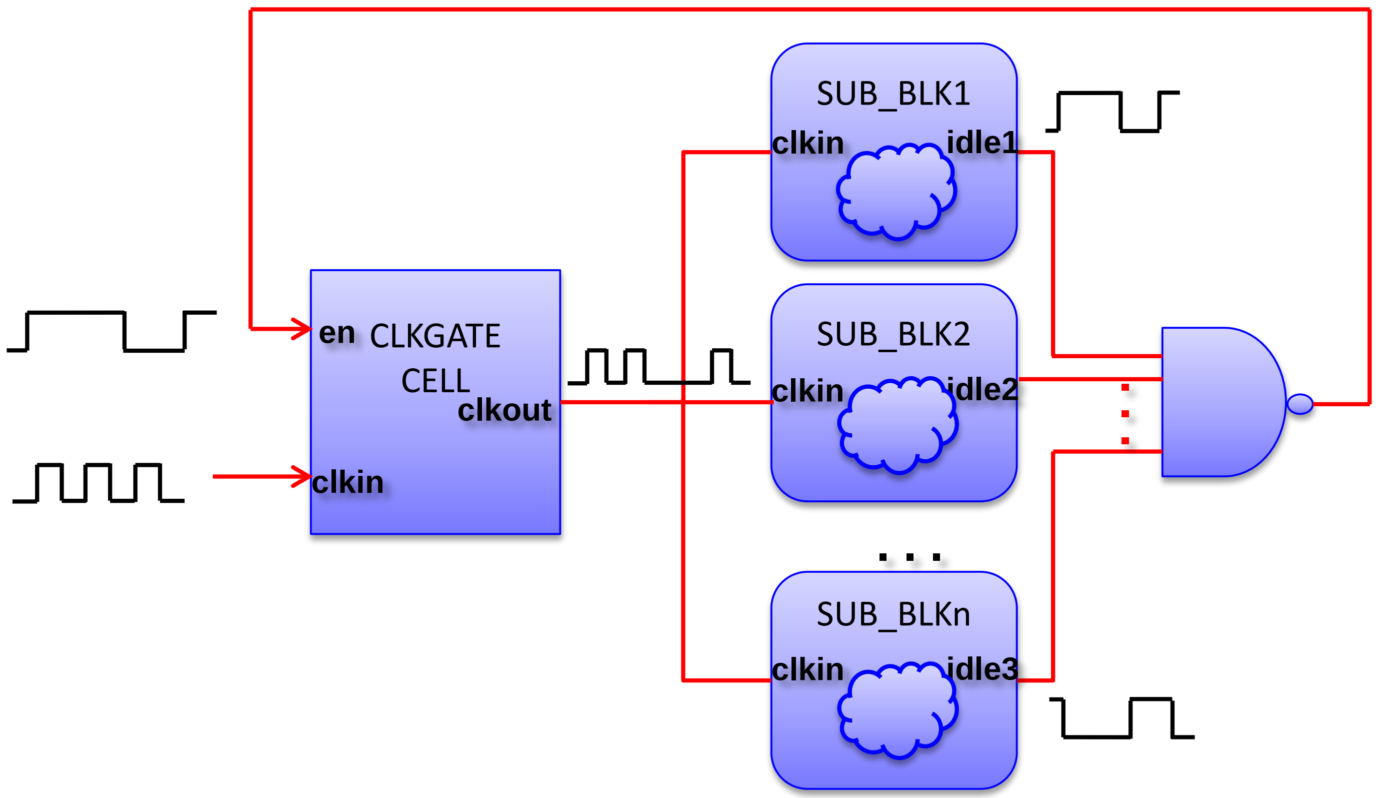
    to detect active states of all components

    and turn off clocks if all are idle



Hardware Controlled Clock Gating Design

- Use automatic clock gating logic to detect active states of all components and turn off clocks if all are idle



Clock Gating Verification Challenges

- It's hard for simulation methods to achieve thorough coverage and automatic checking
  - The CLKGATE's enable is always 1
  - The CLKGATE's enable is always 0
  - The CLKGATE's enable is 0 when it should be 1
  - The CLKGATE's enable is 1 when it should be 0
- Formal technique is a must in such scenarios
  - Formal sequential logic equivalence checking
  - No promising results due to the convergence difficulty and constraint complexity

Verification Methodology

- A systematic clock gating verification methodology is proposed targeting hardware controlled clock gating strategies.
  - Combines structural analysis, formal property auto-generation and proof
- 3-steps verification:
  - Check each clock gate cell's enable input is not *stuck-at-0* or *stuck-at-1*
  - Check number of fan-out flops of each clock gate cell
  - Check cross domain clock gating logic properties and errors
- Use formal verification tool's TCL based user interface:
  - comprehensive design traversal
  - automated assertion generation and proof
  - property creation during runtime

Enable *stuck-at-0/1* Checking

- The hardware clock gating logic uses the equation conceptually as:

```
CLKGATE.enable = ~idle
               = ~( idle₁ & idle₂ & ... & idleₙ )
               = active₁ | active₂ | ... | activeₙ
```

- Clock gating cells' enable should not be always 0 or always 1
- Develop two formal cover properties to detect enable *stuck-at-0* and *stuck-at-1* issues
  - Property en_high: **@(posedge clk_in) (en) ;**
  - Property en_low: **@(posedge clk_in) (!en) ;**
- Bind properties to each CLKGATE cell instance, use formal tool to prove

Enable *stuck-at-0/1* Checking - Results

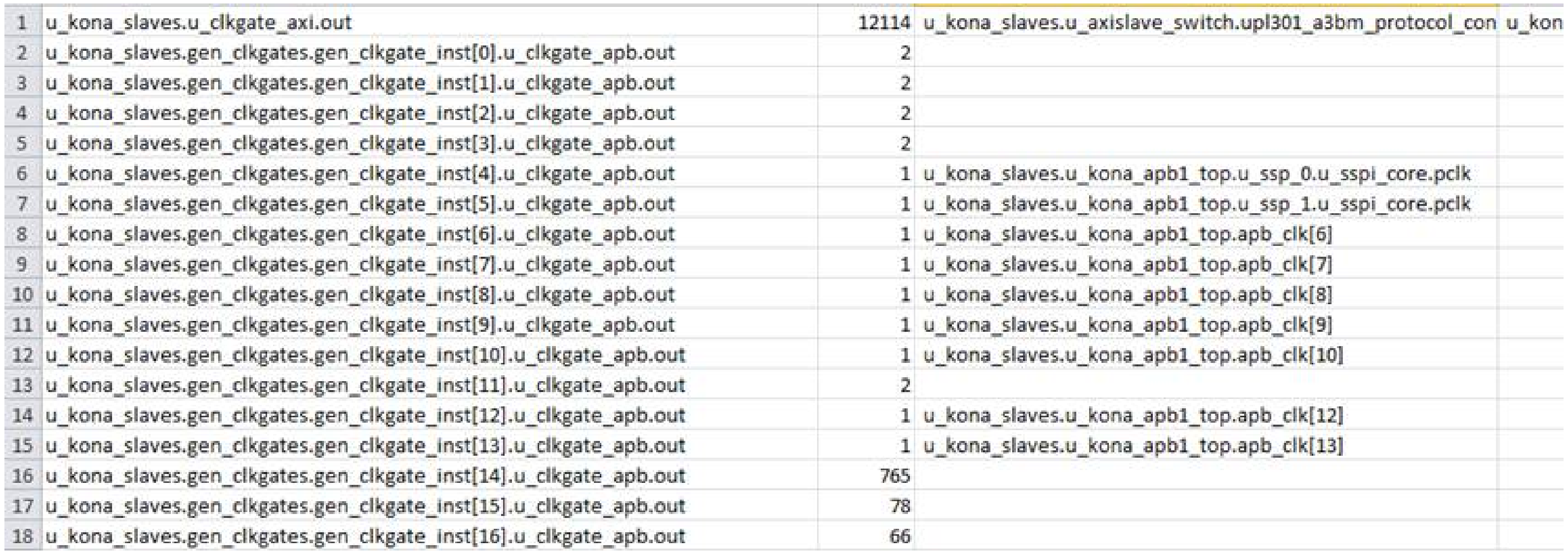
- One CLKGATE.enable stuck-at-0 case
  - Designer tied the clock of one unused block to 0
- Several stuck-at-1 cases/bugs
  - A real bug in a third party IP: design logic is always 1
  - Designer tied enable to 1 feeling auto gating logic is too risky to use
  - Designer tied enable to 1 on purpose to allow clock to be controlled by the upper level

```
// Enable signal for the SE and BE outputs
assign edac_se_en =  edac_se_live_not_zero |        // requests not granted
                     sp_req_set            |        // Request set signal
                     edac_ready;                    // Ready signal

// Ready signal after all bytes were granted by the internal arbiters
assign edac_ready =  edac_se_live_is_zero |         // All requests were granted
                     ~sp_req_r;                     // No request is active
```

Fan-out Flops Checking

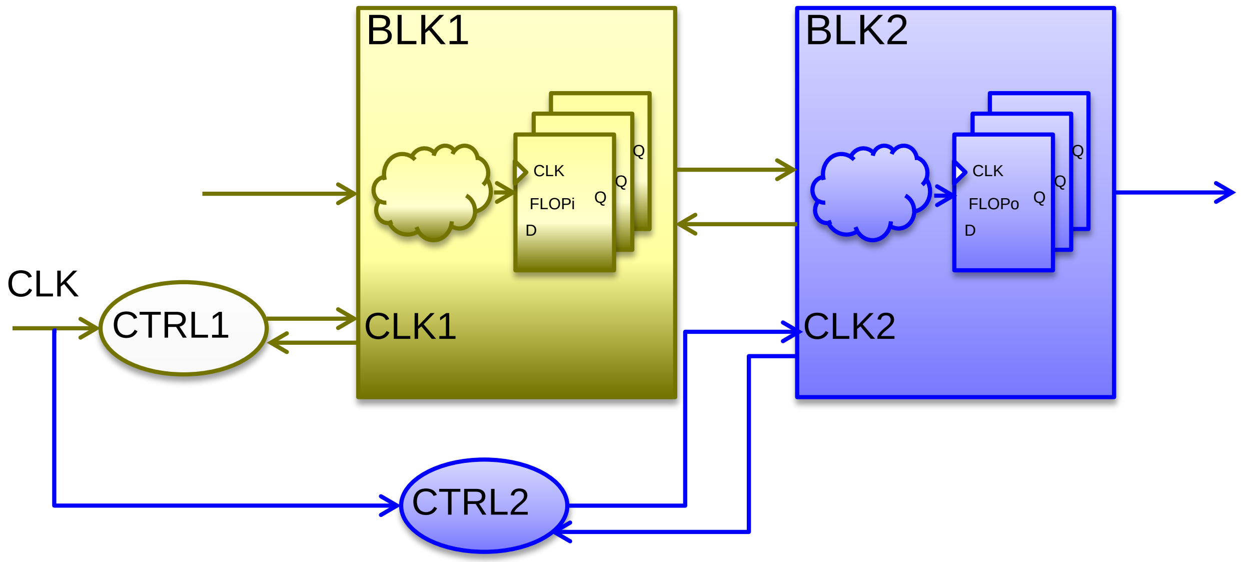
- Some clock gate cells drive very few flip-flops
- Such clock gating logic design is not very efficient
- TCL procedures to analyze the RTL clock tree of each clock gate and to report the number of flip-flops it drives



1	u_kona_slaves.u_clkgate_axi.out	12114	u_kona_slaves.u_axislave_switch.upl301_a3bm_protocol_con	u_kon
2	u_kona_slaves.gen_clkgates.gen_clkgate_inst[0].u_clkgate_apb.out	2		
3	u_kona_slaves.gen_clkgates.gen_clkgate_inst[1].u_clkgate_apb.out	2		
4	u_kona_slaves.gen_clkgates.gen_clkgate_inst[2].u_clkgate_apb.out	2		
5	u_kona_slaves.gen_clkgates.gen_clkgate_inst[3].u_clkgate_apb.out	2		
6	u_kona_slaves.gen_clkgates.gen_clkgate_inst[4].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.u_ssp_0.u_sspi_core.pclk	
7	u_kona_slaves.gen_clkgates.gen_clkgate_inst[5].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.u_ssp_1.u_sspi_core.pclk	
8	u_kona_slaves.gen_clkgates.gen_clkgate_inst[6].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[6]	
9	u_kona_slaves.gen_clkgates.gen_clkgate_inst[7].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[7]	
10	u_kona_slaves.gen_clkgates.gen_clkgate_inst[8].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[8]	
11	u_kona_slaves.gen_clkgates.gen_clkgate_inst[9].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[9]	
12	u_kona_slaves.gen_clkgates.gen_clkgate_inst[10].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[10]	
13	u_kona_slaves.gen_clkgates.gen_clkgate_inst[11].u_clkgate_apb.out	2		
14	u_kona_slaves.gen_clkgates.gen_clkgate_inst[12].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[12]	
15	u_kona_slaves.gen_clkgates.gen_clkgate_inst[13].u_clkgate_apb.out	1	u_kona_slaves.u_kona_apb1_top.apb_clk[13]	
16	u_kona_slaves.gen_clkgates.gen_clkgate_inst[14].u_clkgate_apb.out	765		
17	u_kona_slaves.gen_clkgates.gen_clkgate_inst[15].u_clkgate_apb.out	78		
18	u_kona_slaves.gen_clkgates.gen_clkgate_inst[16].u_clkgate_apb.out	66		

Cross Domain Property Checking

- Two scenarios that a bug can happen in the RTL design
  - *BLK1* is passing transaction information to *BLK2* but *CLK2* is gated
  - *CLK1* getting gated prematurely when a transaction is not completed yet in *BLK1*



Basic Strategy

- Two properties are created to catch those bugs
  - Property 1:

```
not ($changed(FLOPi.D_on_CLK1) && $changed(CLK1) ##1
     $stable(CLK1) && $changed(CLK2) ##1 $changed(CLK2))
```

  - Property 2:

```
not ($changed(FLOPi.D_on_CLK1) ##1 $changed(CLK1) &&
     $changed(CLK2) ##1 $changed(CLK1) && $stable(CLK2))
```

- TCL procedures and scripts are developed to:
  - Do structural analysis of the design using JasperGold APIs
  - Extract the structural relationship between the flops on CLK1 domain and the flops on CLK2 domain
  - Automatically generate properties on the flip-flops of interests

Illustration of Property 1

```
not ($changed(FLOPi.D_on_CLK1) && $changed(CLK1) ##1
     $stable(CLK1) && $changed(CLK2) ##1 $changed(CLK2))
```

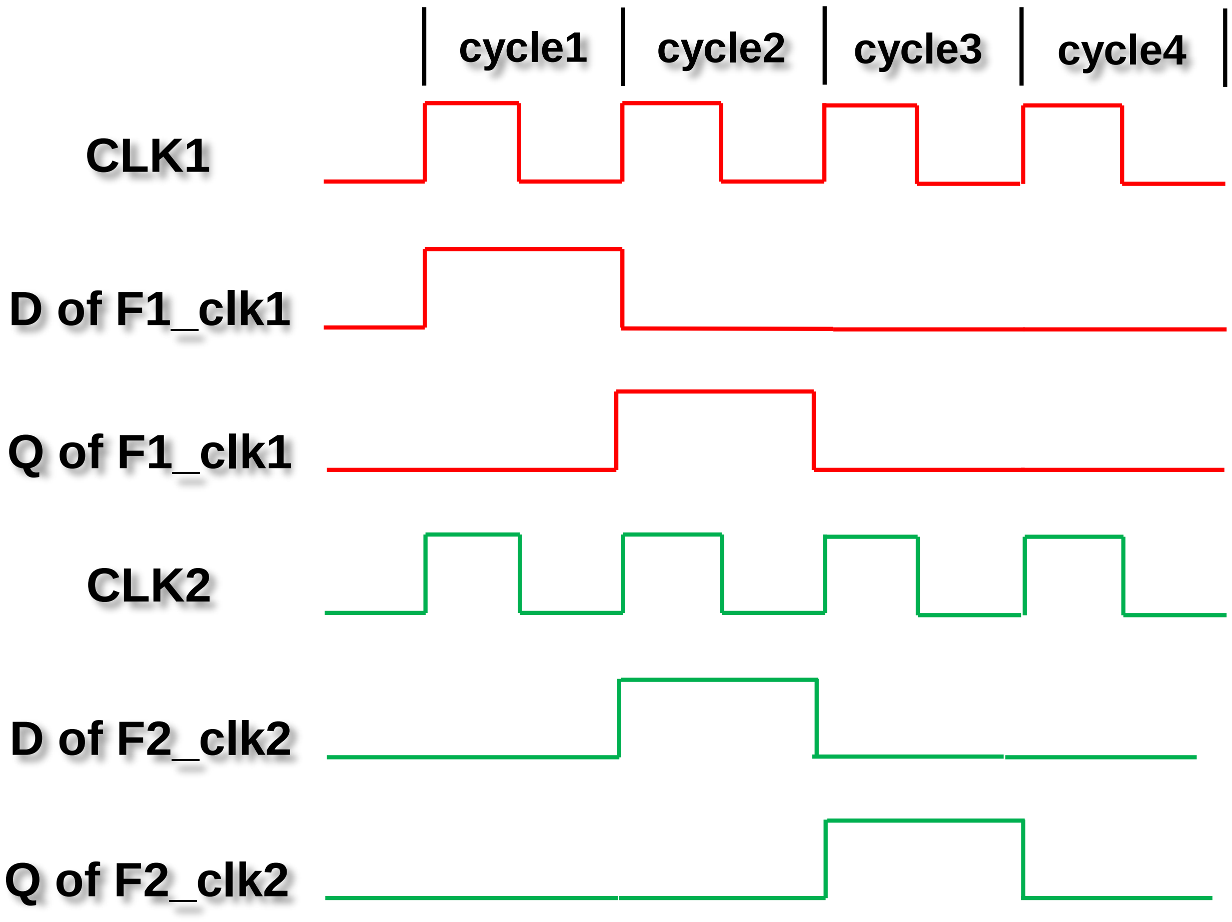


Illustration of Property 1

```
not ($changed(FLOPi.D_on_CLK1) && $changed(CLK1) ##1
     $stable(CLK1) && $changed(CLK2) ##1 $changed(CLK2))
```

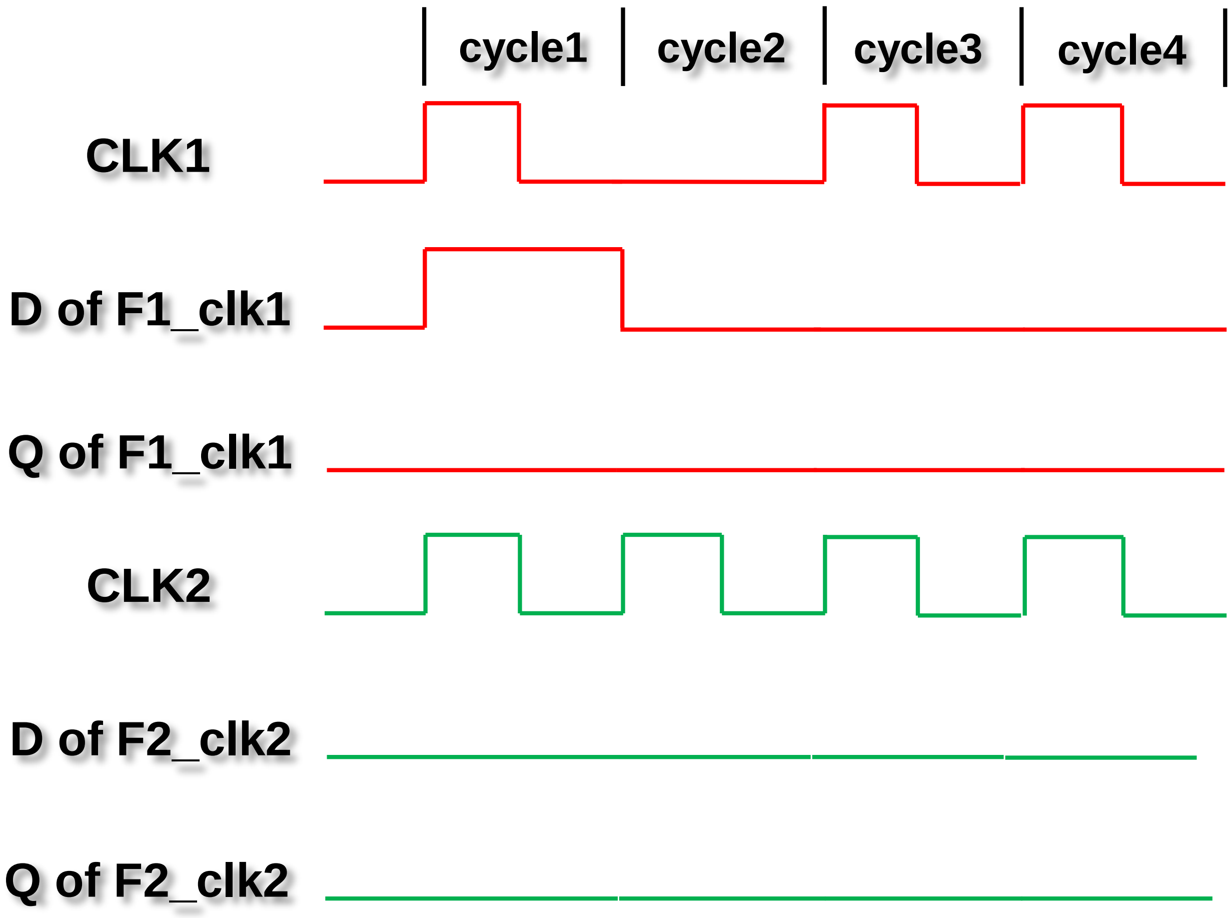
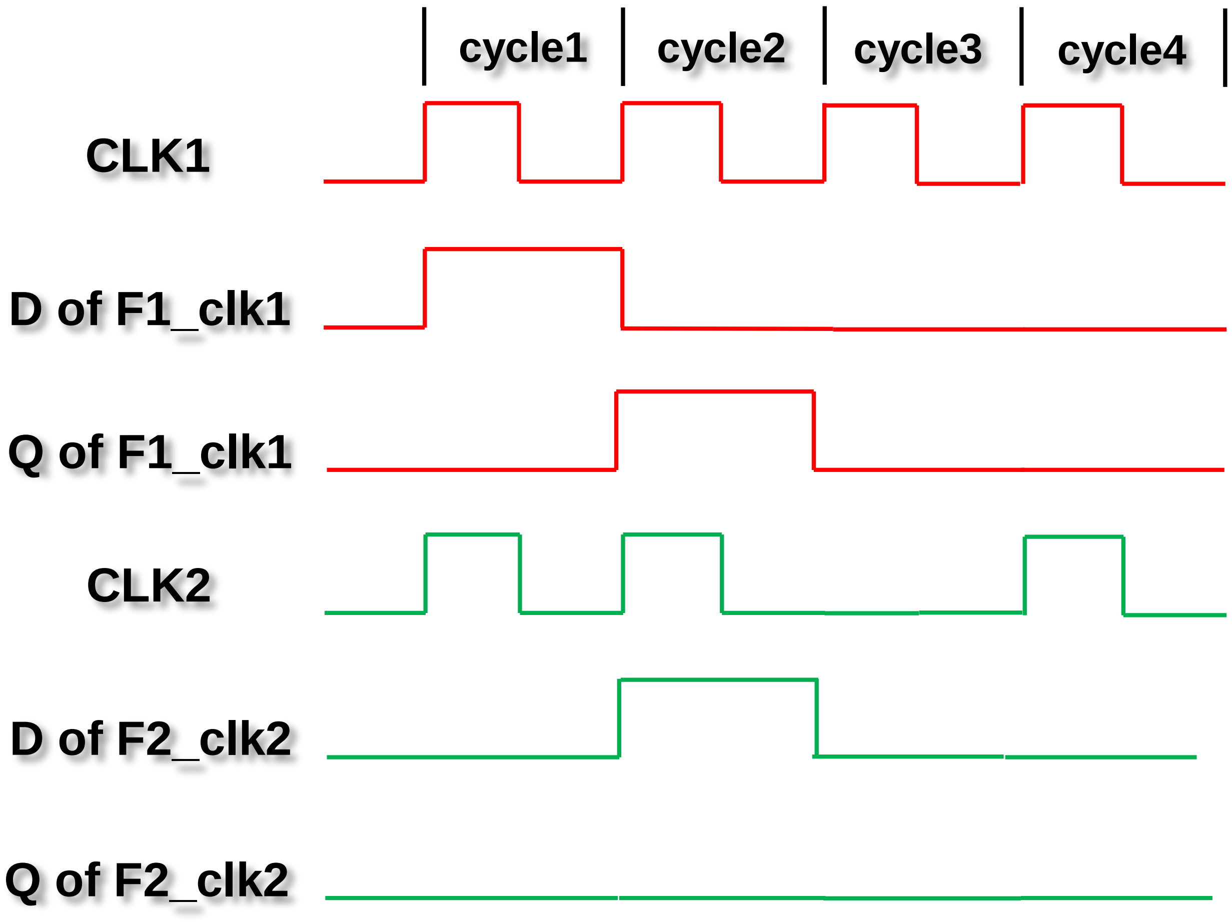


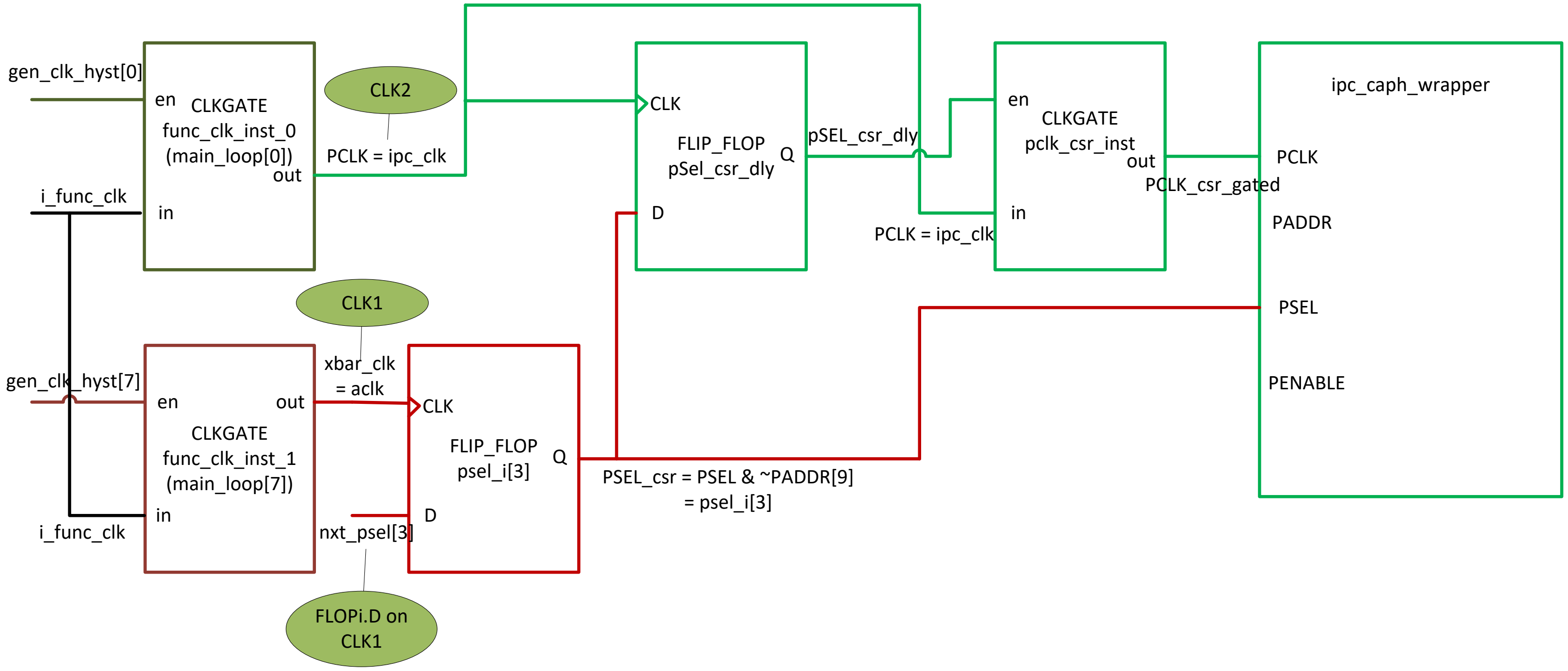
Illustration of Property 2

```
not ($changed(FLOPi.D_on_CLK1) ##1 $changed(CLK1) &&
     $changed(CLK2) ##1 $changed(CLK1) && $stable(CLK2))
```

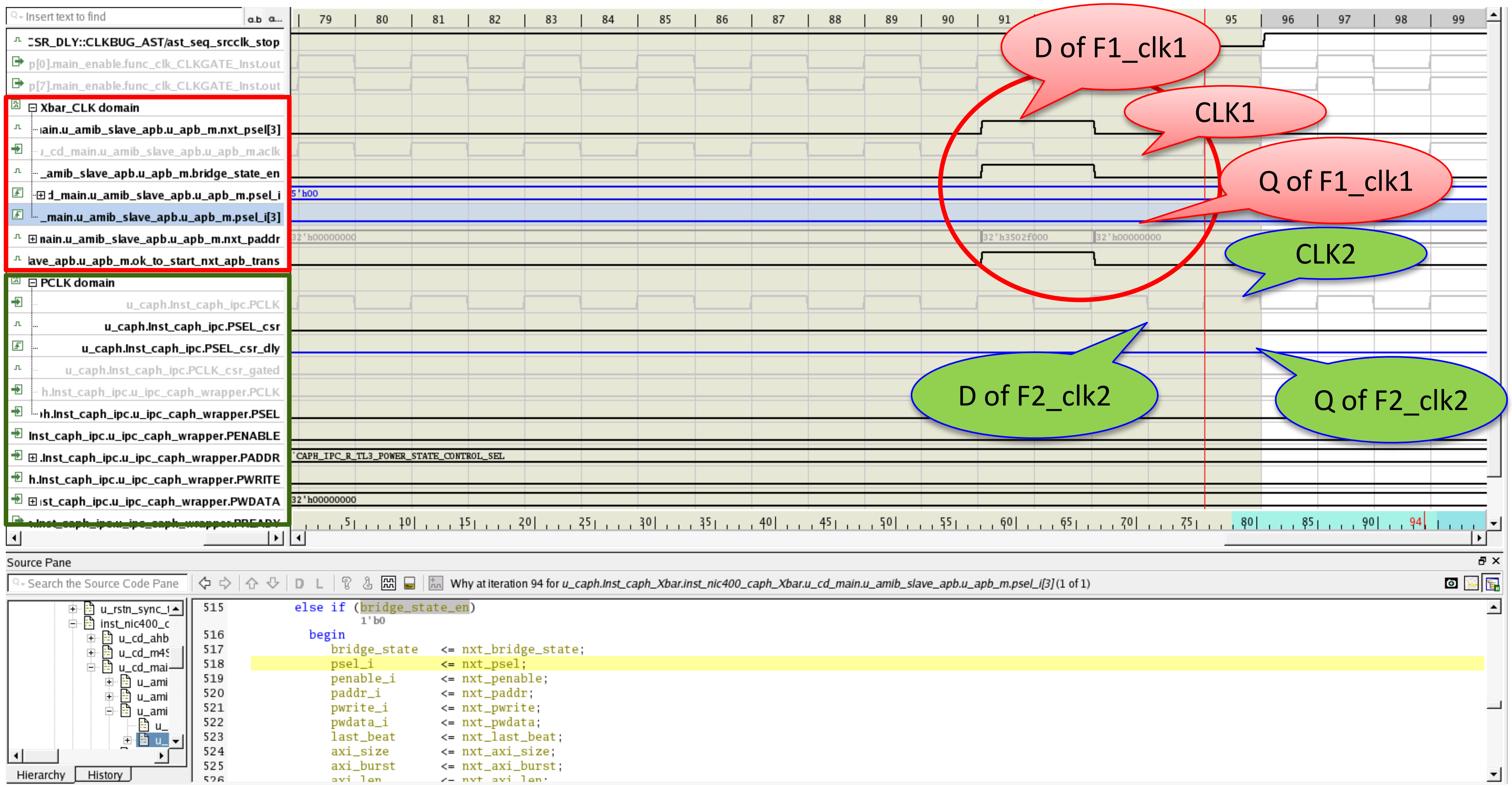


Cross Domain Clock Gating Logic Checking Results

- One complex RTL bug was found
  - Transactions from CLK1 domain could not reach CLK2 domain due to the wrongly gated clocks
  - Two scenarios violating the two properties were generated to guide simulation and bug fixing



Counter Example 1

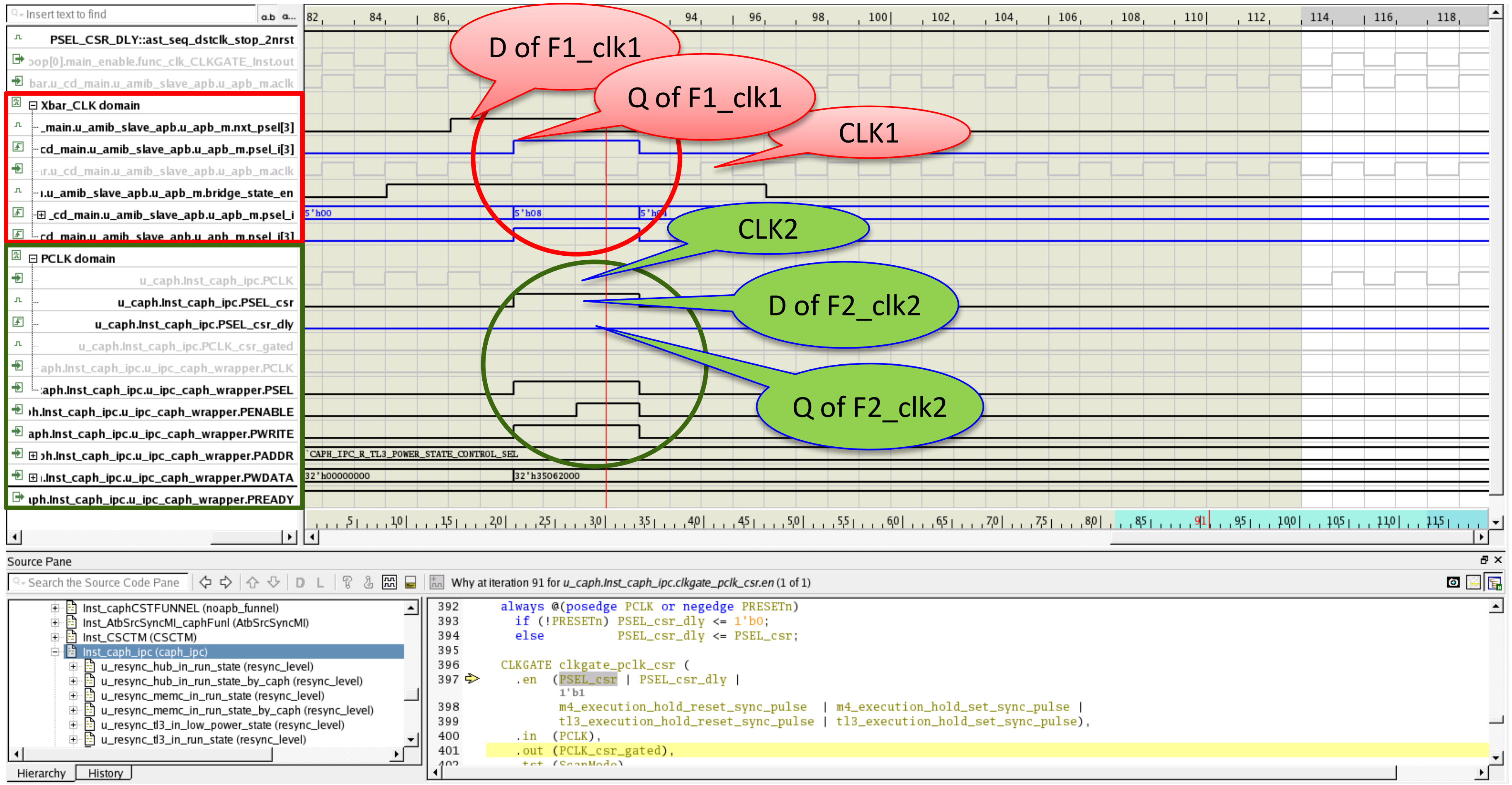


Insert text to find
ɔSR_DLY::CLKBUG_AST/ast_seq_srcclk_stop
p[0].main_enable.func_clk_CLKGATE_Inst.out
p[7].main_enable.func_clk_CLKGATE_Inst.out
Xbar_CLK domain
ıain.u_amib_slave_apb.u_apb_m.nxt_psel[3]
ı_cd_main.u_amib_slave_apb.u_apb_m.aclk
_amib_slave_apb.u_apb_m.bridge_state_en
d_main.u_amib_slave_apb.u_apb_m.psel_i
_main.u_amib_slave_apb.u_apb_m.psel_i[3]
nain.u_amib_slave_apb.u_apb_m.nxt_paddr
lave_apb.u_apb_m.ok_to_start_nxt_apb_trans
PCLK domain
u_caph.Inst_caph_ipc.PCLK
u_caph.Inst_caph_ipc.PSEL_csr
u_caph.Inst_caph_ipc.PSEL_csr_dly
u_caph.Inst_caph_ipc.PCLK_csr_gated
h.Inst_caph_ipc.u_ipc_caph_wrapper.PCLK
ɔh.Inst_caph_ipc.u_ipc_caph_wrapper.PSEL
Inst_caph_ipc.u_ipc_caph_wrapper.PENABLE
.Inst_caph_ipc.u_ipc_caph_wrapper.PADDR
h.Inst_caph_ipc.u_ipc_caph_wrapper.PWRITE
ıst_caph_ipc.u_ipc_caph_wrapper.PWDATA
5'h00
32'h00000000
32'h3502f000
32'h00000000
`CAPH_IPC_R_TL3_POWER_STATE_CONTROL_SEL
32'h00000000
79 80 81 82 83 84 85 86 87 88 89 90 91 95 96 97 98 99
D of F1_clk1
CLK1
Q of F1_clk1
CLK2
D of F2_clk2
Q of F2_clk2
Source Pane
Search the Source Code Pane
Hierarchy
History

Why at iteration 94 for *u_caph.Inst_caph_Xbar.inst_nic400_caph_Xbar.u_cd_main.u_amib_slave_apb.u_apb_m.psel_i[3]* (1 of 1)

```
        else if (bridge_state_en)
                  1'b0
          begin
             bridge_state    <= nxt_bridge_state;
             psel_i          <= nxt_psel;
             penable_i       <= nxt_penable;
             paddr_i         <= nxt_paddr;
             pwrite_i        <= nxt_pwrite;
             pwdata_i        <= nxt_pwdata;
             last_beat       <= nxt_last_beat;
             axi_size        <= nxt_axi_size;
             axi_burst       <= nxt_axi_burst;
```

Counter Example 2



PSEL_CSR_DLY::ast_seq_dstclk_stop_2nrst
oop[0].main_enable.func_clk_CLKGATE_Inst.out
bar.u_cd_main.u_amib_slave_apb.u_apb_m.aclk
Xbar_CLK domain
_main.u_amib_slave_apb.u_apb_m.nxt_psel[3]
cd_main.u_amib_slave_apb.u_apb_m.psel_i[3]
r.u_cd_main.u_amib_slave_apb.u_apb_m.aclk
.u_amib_slave_apb.u_apb_m.bridge_state_en
_cd_main.u_amib_slave_apb.u_apb_m.psel_i
cd_main.u_amib_slave_apb.u_apb_m.psel_i[3]
PCLK domain
u_caph.Inst_caph_ipc.PCLK
u_caph.Inst_caph_ipc.PSEL_csr
u_caph.Inst_caph_ipc.PSEL_csr_dly
u_caph.Inst_caph_ipc.PCLK_csr_gated
aph.Inst_caph_ipc.u_ipc_caph_wrapper.PCLK
aph.Inst_caph_ipc.u_ipc_caph_wrapper.PSEL
h.Inst_caph_ipc.u_ipc_caph_wrapper.PENABLE
aph.Inst_caph_ipc.u_ipc_caph_wrapper.PWRITE
h.Inst_caph_ipc.u_ipc_caph_wrapper.PADDR
.Inst_caph_ipc.u_ipc_caph_wrapper.PWDATA
ph.Inst_caph_ipc.u_ipc_caph_wrapper.PREADY

5'h00 5'h08 5'h0

`CAPH_IPC_R_TL3_POWER_STATE_CONTROL_SEL

32'h00000000 32'h35062000

Why at iteration 91 for *u_caph.Inst_caph_ipc.clkgate_pclk_csr.en* (1 of 1)

Inst_caphCSTFUNNEL (noapb_funnel)
Inst_AtbSrcSyncMl_caphFunl (AtbSrcSyncMl)
Inst_CSCTM (CSCTM)
Inst_caph_ipc (caph_ipc)
u_resync_hub_in_run_state (resync_level)
u_resync_hub_in_run_state_by_caph (resync_level)
u_resync_memc_in_run_state (resync_level)
u_resync_memc_in_run_state_by_caph (resync_level)
u_resync_tl3_in_low_power_state (resync_level)
u_resync_tl3_in_run_state (resync_level)

```
always @(posedge PCLK or negedge PRESETn)
  if (!PRESETn) PSEL_csr_dly <= 1'b0;
  else          PSEL_csr_dly <= PSEL_csr;

CLKGATE clkgate_pclk_csr (
  .en  (PSEL_csr | PSEL_csr_dly |
        1'b1
        m4_execution_hold_reset_sync_pulse  | m4_execution_hold_set_sync_pulse |
        t13_execution_hold_reset_sync_pulse | t13_execution_hold_set_sync_pulse),
  .in  (PCLK),
  .out (PCLK_csr_gated),
  .tst (ScanMode)
```

Summary

- To maximize dynamic power reduction, our SoC design adopted an aggressive clock gating design methodology using manually inserted ICG as close as possible to the root of the clock tree
- Low-power verification can be inherently complex and the formal technique is a must in such scenarios
- A unique and systematic formal methodology combining structural analysis, automatic property generation and proof was presented for complex clock gating design verification
- Bugs found were complex in nature and hard to find in a controlled way using simulation

Clock-Enable Control Type	Verification Methodology
Hardware Controlled	Check enable inputs for all clock gate cells are not stuck-at-0 or stuck-at-1
	Analyze number of flip-flops that each clock gate drives
	Do structural fan-in and fan-out analysis of each clock-gated flip-flop and automatically generate properties to check cross-domain errors

Thank You!