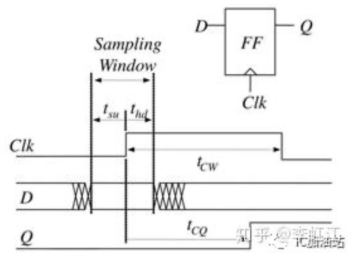
Teacher LEE's lecture links

- CDC_1_Introduction
- CDC_2_2-Flop-Synchronizer
- CDC_3_Pulse-Synchronizer
- CDC_4_Multi-bit_CDC
- CDC_5_Asynchronous_FIFO_Part1
- CDC_6_Asynchronous_FIFO_Part2
- CDC_7_Synopsys-Spyglass

CDC — Clock Domain Crossing

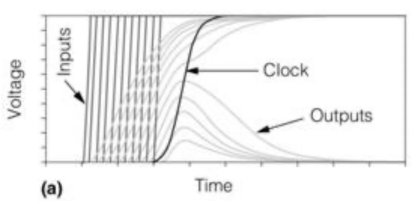
DFF Timing

- setup time: Required stable time for D signal before clk arrives
- hold time: Required stable time for D signal after clk arrives
- clk-to-q time: The time from clk to Q if setup/hold time criteria satisfied



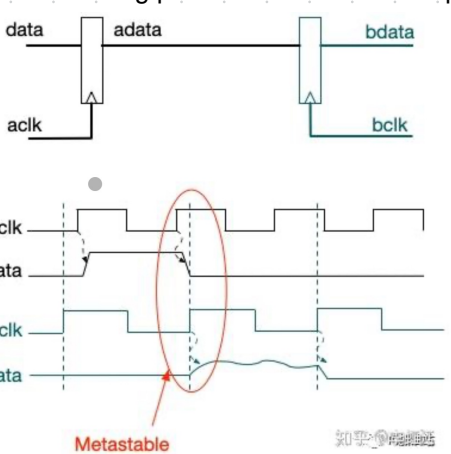
Metastability

- D signal change inside setup + hold time window
- clk-to-q behavior change (longer time to stabilize)

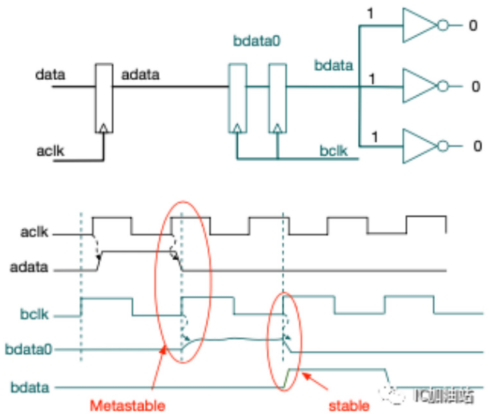


CDC causing metastability

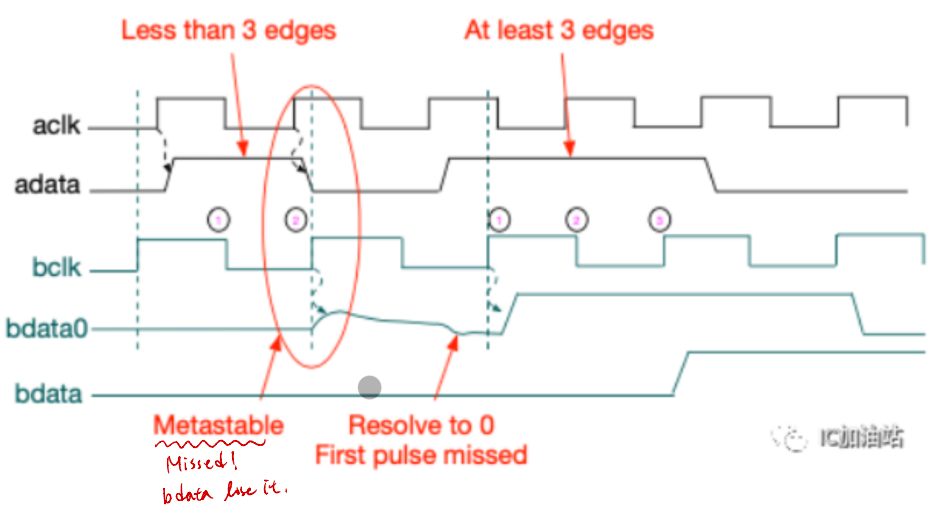
- Different clk domain causing potential violation to setup/hold condition



2-Flop-Synchronizer

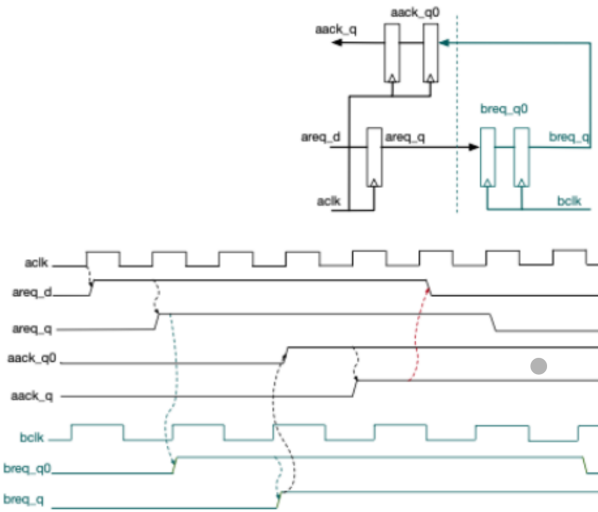


- Metastability can't be guarantee 100% solved!!!!!
  - D change closer to clk edge => Metastability stabilize time increase
  - Can't guarantee sufficient clk cycles for Q to stabilize
- MTBF (Mean time between failure)
  - More flop stages => MTBF decrease significantly
  - Design product with sufficient MTBF probability is enough
- 3 edge rule for double-flop synchronizer
  - Pulse miss case: input pulse too short for second clk to capture
  - Input pulse longer than 3 consecutive output clk rising/falling edge

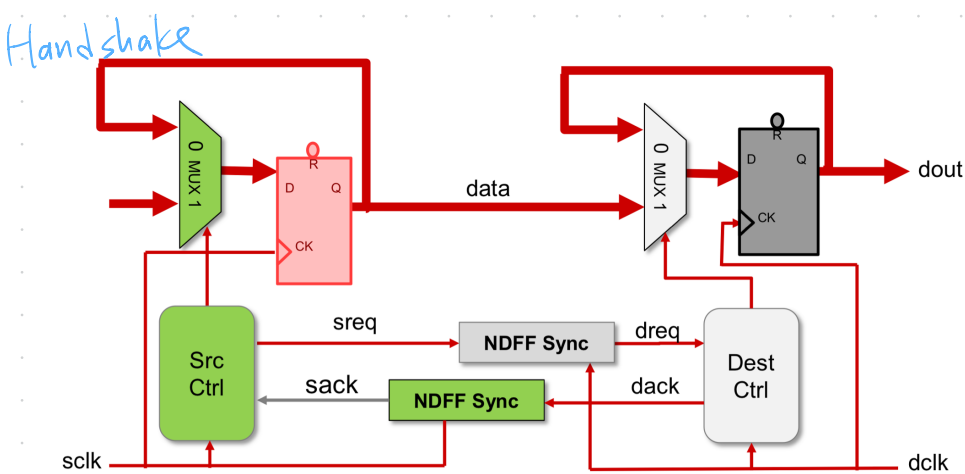


- bclk frequency should be at least 1.5 times of aclk frequency to ensure single pulse input transmitted from A domain to B domain

- Handshake feedback req-ack pair to ensure signal properly transmitted

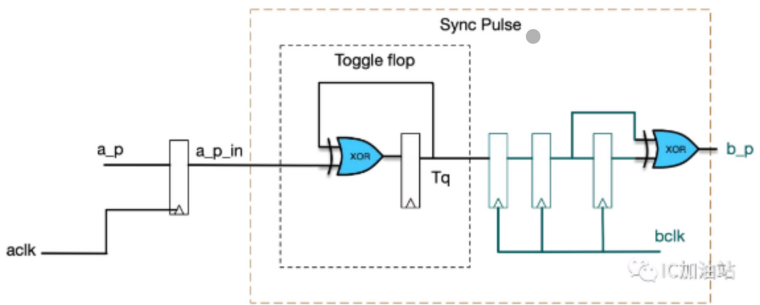


- Random resolve time for double-flop synchronizer: 1 or 2 cycles
- Input of synchronizer should be DFF output!!!!! (Can't be comb output)
- How to read single pulse? => Pulse synchronizer

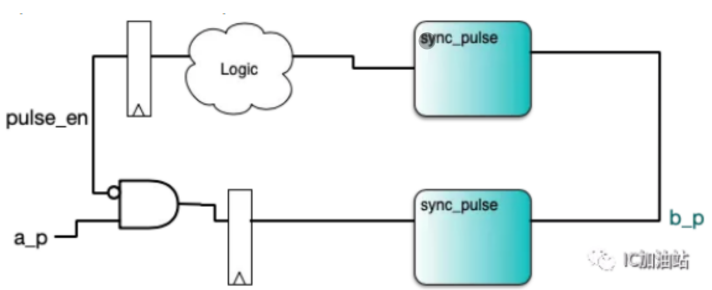


Pulse Synchronizer

- Transfer 1 aclk pulse from A domain to 1 bclk pulse in B domain
  - Direct 2-flop-synchronizer can't guarantee this behavior
  - Used on behavior flag (counter add one flag / CE, WE flag for memory, etc.)
- Toggle Flop
  - Pulse => Level => 2-flop-synchronizer => Level => Pulse
  - Toggle flop triggers on pulse signal

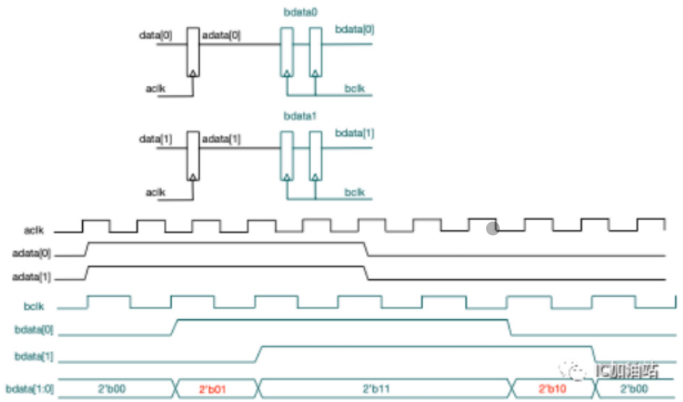


- 3 edge rule for pulse synchronizer
  - Ensure 2-flop-synchronizer get the level signal
  - Gap between two aclk pulses must be > 3 bclk edges
  - Use feedback to limit aclk request frequency (inefficient? => use FIFO!)

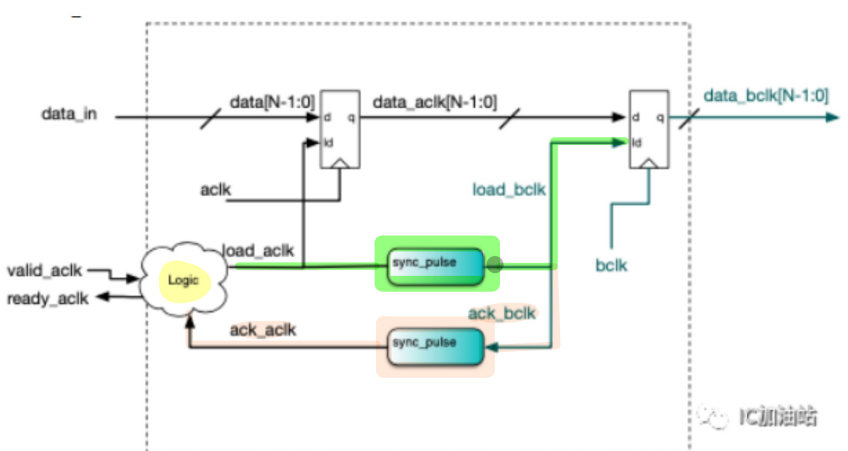


Multi-bit CDC

- Multi-bit mismatch through synchronizer

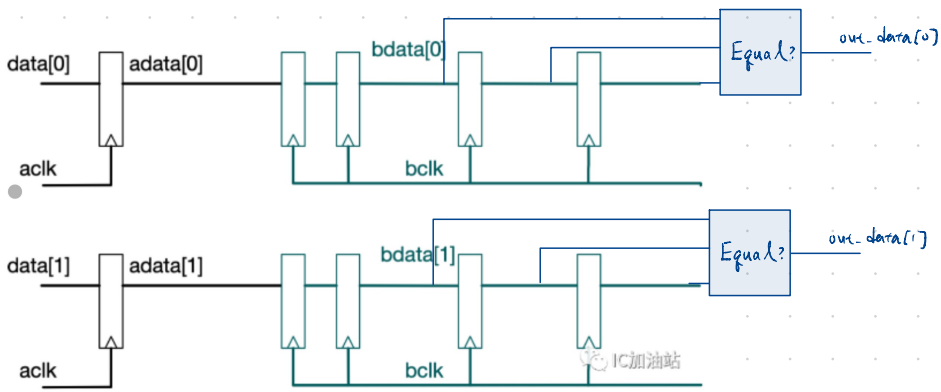


- Solution 1: 1-bit Load flag through synchronizer (1-entry FIFO)
  - Load flag
    - High when data_aclk is stable
    - 1-cycle pulse => data_bclk capture 1 time only
    - Feedback ack signal for next load flag



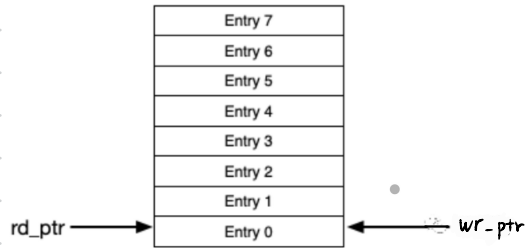
  - Used on non-high speed CDC transfer

- Solution 2: Multi-stage FF stabilize detection
  - Compare multi-stage FF value
    - All stage value equal => output bit is valid
  - Multi-bit input should be stable for long enough (can't be switching frequently)
  - Large area cost when bit number high



- Solution 3: FIFO?

Asynchronous FIFO prerequisite



- read_ptr & write_ptr => multi-bit CDC issue
  - Push: write_ptr + 1
  - Pop: read_ptr + 1
- Push & Pop locate in different clk domain
  - FIFO full => Push invalid
  - FIFO empty => Pop invalid
- Gray Code
  - 2 neighboring Gray code has 1 bit different
  - When Nth bit 0 => 1
    - Bit > N => the same
    - Bit < N => mirror to transition point → e.g. 4'b0100 ≠ 4'b1100

4'd0	4'b0000	4'b0000
4'd1	4'b0001	4'b0001
4'd2	4'b0010	4'b0011
4'd3	4'b0011	4'b0010
4'd4	4'b0100	4'b0110
4'd5	4'b0101	4'b0111
4'd6	4'b0110	4'b0101
4'd7	4'b0111	4'b0100
4'd8	4'b1000	4'b1100
4'd9	4'b1001	4'b1101
4'd10	4'b1010	4'b1111
4'd11	4'b1011	4'b1110
4'd12	4'b1100	4'b1010
4'd13	4'b1101	4'b1011
4'd14	4'b1110	4'b1001
4'd15	4'b1111	4'b1000

mirrored

same

N

    - XOR to convert between Binary <=> Gray
      - g(n) = b(n+1) ^ b(n) (b(n+1) = 0 if doesn't exist)
      - b(n) = b(n+1) ^ g(n) (compute from nth to 0th bit)

e.g.

$g(3) = b(3) \wedge 0$
$g(2) = b(2) \wedge b(3)$
$g(1) = b(1) \wedge b(2)$
$g(0) = b(0) \wedge b(1)$

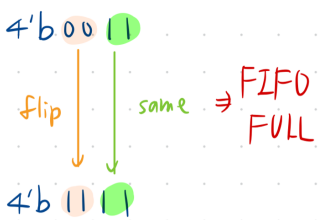
$b(3) = g(3)$
$b(2) = b(3) \wedge g(2)$
$b(1) = b(2) \wedge g(1)$
$b(0) = b(1) \wedge g(0)$

- Non Glitch transition
  - Binary may switch multi-bit at once => Miss-match glitch value through sync.
  - Gray switch 1 bit only => No glitch/error after 2-flop-synchronizer
- Full / Empty detection using Gray code
  - For 2^n FIFO entry => use n+1 bit Gray code address
  - FIFO empty => rd_ptr == wr_ptr
  - FIFO full => (wr_ptr[n:n-1] == ~rd_ptr[n:n-1]) && (wr_ptr[n-2:0] == rd_ptr[n-2:0])

```
assign full = (write_ptr_gray[N:N-1] == ~read_ptr_gray[N:N-1])
            &&(write_ptr_gray[N-2:0] == read_ptr_gray[N-2:0]);
```

eg. 8 entries, n=3, use 3+1 bit address

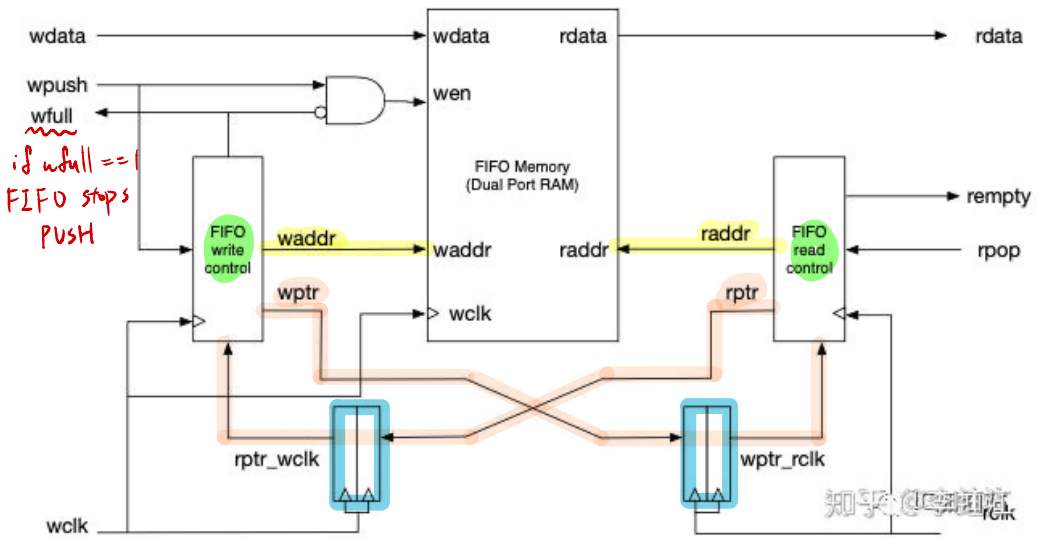
4'd0	4'b0000	4'b0000
4'd1	4'b0001	4'b0001
4'd2	4'b0010	4'b0011 ← rd_ptr
4'd3	4'b0011	4'b0010
4'd4	4'b0100	4'b0110
4'd5	4'b0101	4'b0111
4'd6	4'b0110	4'b0101
4'd7	4'b0111	4'b0100
4'd8	4'b1000	4'b1100
4'd9	4'b1001	4'b1101
4'd10	4'b1010	4'b1111 ← wr_ptr
4'd11	4'b1011	4'b1110
4'd12	4'b1100	4'b1010
4'd13	4'b1101	4'b1011
4'd14	4'b1110	4'b1001
4'd15	4'b1111	4'b1000



- Question: In case [Slow clk Gray code] => [Fast clk Gray code], will there occurs multi-bit metastability? (Current Gray and next Gray has multi-bit difference)
  - No multi-bit metastability occurs at the same time!!!!! => No glitch value!!!
  - For destination clk edge => at most 1 bit toggle at a time, at most 1 bit occurs metastability issue

Asynchronous FIFO

- Structure: Control + Dual-port RAM
  - Binary: waddr, radar
  - Gray: wptr, rptr (through 2-flop-synchronizer)
  - Gray <=> Binary conversion inside FIFO read/write control



- FIFO memory choice
  - Vendor compile memory => typically used on > 2k bits
    - area/power efficient
    - technology-specific / harder to customize
  - Flop array => used on < 2k bits
    - larger area/power
    - Better cusomizability / easier to debug

- Write control: wptr(gray), waddr(binary), wfull

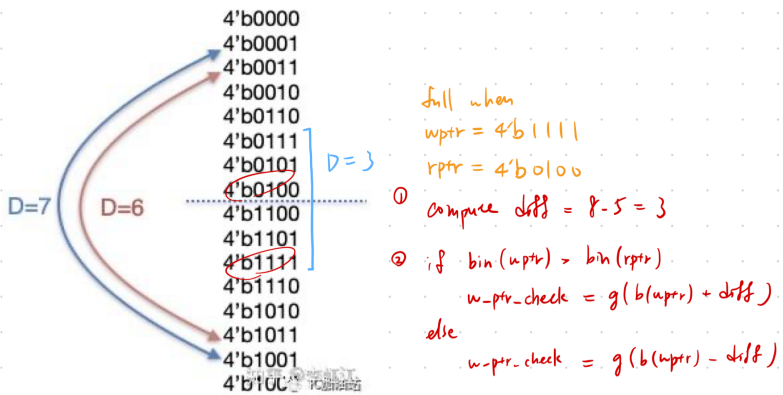
```
module wptr_full  #(parameter ADDRSIZE = 4)
        (output reg                wfull,
         output     [ADDRSIZE-1:0] waddr,
         output reg [ADDRSIZE  :0] wptr,
         input      [ADDRSIZE  :0] rptr_wclk,
input
wpush, wclk, wrst_n);
reg  [ADDRSIZE:0] wbin;
wire [ADDRSIZE:0] wgraynext, wbinnext;
// GRAYSTYLE2 pointer
always @(posedge wclk or negedge wrst_n)
  if (!wrst_n) {wbin, wptr} <= 0;
  else         {wbin, wptr} <= {wbinnext, wgraynext};
// Memory write-address pointer (okay to use binary to address memory)
assign waddr = wbin[ADDRSIZE-1:0];
assign wbinnext  = wbin + (wpush & ~wfull);
assign wgraynext = (wbinnext>>1) ^ wbinnext;
//------------------------------------------------------------------
// Simplified version of the three necessary full-tests:
// assign wfull_val=((wgnext[ADDRSIZE] !=wq2_rptr[ADDRSIZE] ) &&
//                   (wgnext[ADDRSIZE-1]  !=wq2_rptr[ADDRSIZE-1]) &&
// (wgnext[ADDRSIZE-2:0]==wq2_rptr[ADDRSIZE-2:0]));
//------------------------------------------------------------------
assign wfull_val = (wgraynext=={~rptr_wclk[ADDRSIZE:ADDRSIZE-1],
                                rptr_wclk[ADDRSIZE-2:0]});
always @(posedge wclk or negedge wrst_n)
  if (!wrst_n) wfull  <= 1'b0;
    else
        wfull  <= wfull_val;
endmodule
```


- Read control: rptr(gray), raddr(binary), rempty

```
module rptr_empty #(parameter ADDRSIZE = 4)
        (output reg                rempty,
         output     [ADDRSIZE-1:0] raddr,
         output reg [ADDRSIZE  :0] rptr,
         input      [ADDRSIZE  :0] wptr_rclk,
         input                     rpop, rclk, rrst_n);
  reg  [ADDRSIZE:0] rbin;
  wire [ADDRSIZE:0] rgraynext, rbinnext;
  //---------------------
  // GRAYSTYLE2 pointer
  //---------------------
  always @(posedge rclk or negedge rrst_n)
    if (!rrst_n) {rbin, rptr} <= 0;
    else         {rbin, rptr} <= {rbinnext, rgraynext};
  // Memory read-address pointer (okay to use binary to address memory)
  assign raddr     = rbin[ADDRSIZE-1:0];
  assign rbinnext  = rbin + (rpop & ~rempty);
  assign rgraynext = (rbinnext>>1) ^ rbinnext;
//---------------------------------------------------------------
// FIFO empty when the next rptr == synchronized wptr or on reset
//---------------------------------------------------------------
assign rempty_val = (rgraynext == wptr_rclk);
  always @(posedge rclk or negedge rrst_n)
    if (!rrst_n) rempty <= 1'b1;
    else         rempty <= rempty_val;
endmodule
```

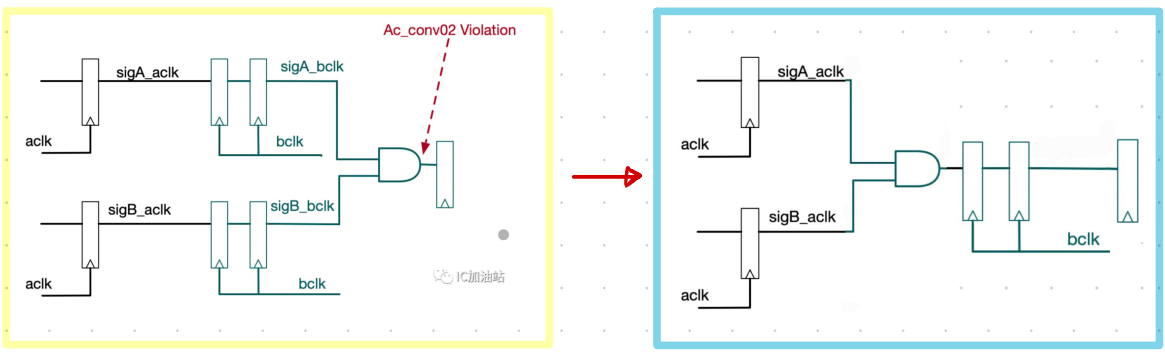
- Pseudo full/empty
  - Scenario: wclk faster than rclk, will write domain push when FIFO is full? NO!!!
    - Pseudo Full: wfull flag will pull up when FIFO is in pseudo full stage (rptr hasn't update through synchronizer)
  - Scenario: rclk faster than wclk, will read domain pop when FIFO is empty? NO!!!
    - Pseudo Empty: rempty will pull up when FIFO is in pseudo empty stage (wptr hasn't update through synchronizer)

- Designing a FIFO with Non-Power-of-Two Depth
  - Use Gray code symmetry property
  - wfull flag logic changed

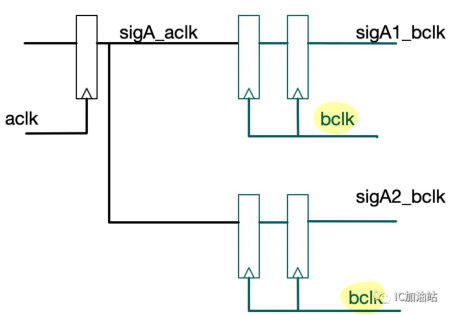


Synopsys Spyglass CDC [structure check] violation

- **Ac_unsync01:** Single-bit CDC not synchronized
- **Ac_unsync02:** Multi-bit CDC not synchronized
- **Ac_conv02:** Multi-bit CDC mismatch issue
  - e.g. sigA_bclk and sigB_bclk might resolve at different time
  - Solve: first do the logic then feed through the synchronizer



- **Ac_coherency06:** Signal synchronized in the same clk domain multiple times
  - Inefficient area / Different resolve time with the same net behavior



- **Ac_glitch02:** Synchronizer input is from comb output, not from DFF output