Use of CDC-Jitter-Modeling in Clock-Domain-Crossing-Circuits in RTL Design Phase

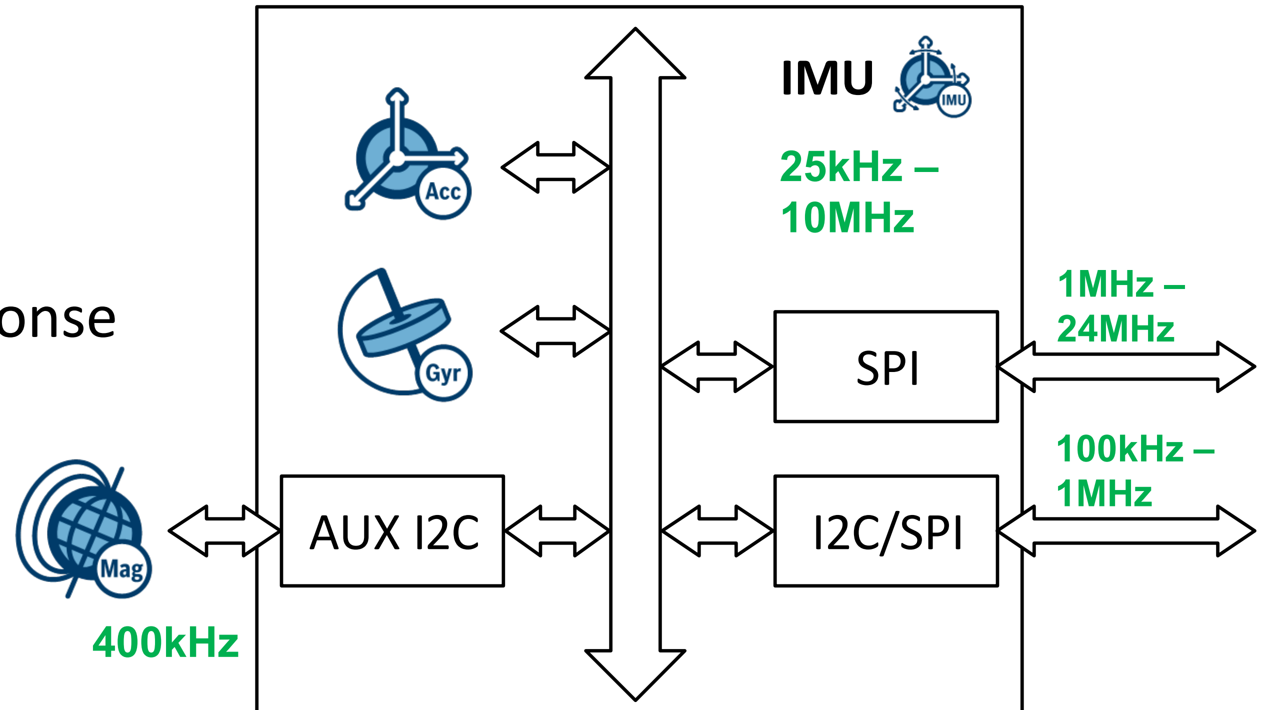
Jan Hayek, Jochen Neidhardt, Robert Richter
Bosch Sensortec GmbH

BOSCH

Motivation

given typical Inertial Measurement Unit (IMU):

- internal clock frequency not fixed
- external clock frequency not fixed
- extensive clock gating
  due low power requirements
- tough constraints for interface response



leads to:

- known good synchronization
  circuits do not fit in all cases
- lots of custom circuits are created …

© Accellera Systems Initiative

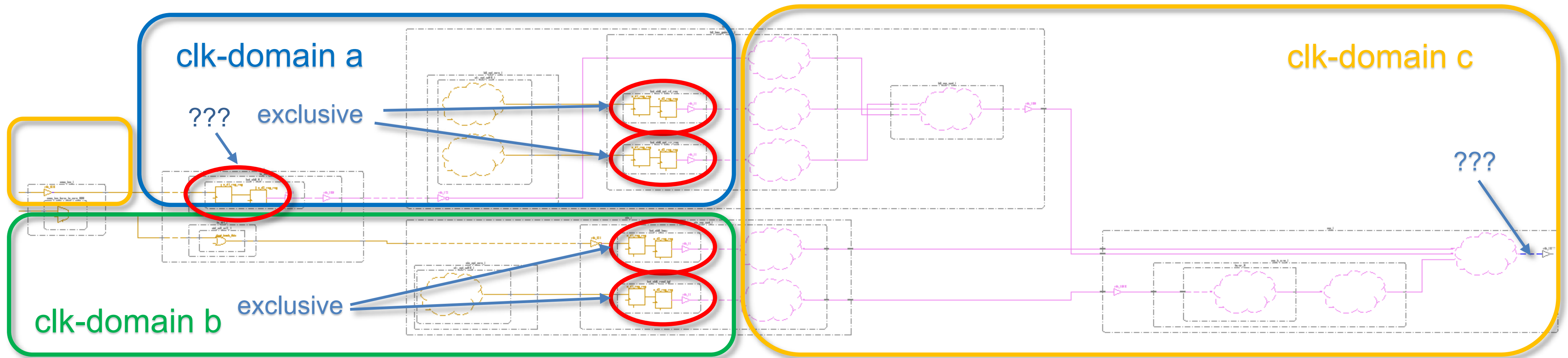
Typical CDC-tasks

task	effort	comment
check for synchronization on asynchronous input	low (CDC-tool)	constraining (sdc) needed for backend anyway
asynchronous signals must be glitch-free	low / medium (CDC-tool)	might need don't touch glitch free logic if not registered (review)
check common data synchronization structure	low / medium (CDC-tool)	usage of CDC-library recommended
check proper reset synchronization	low / medium (CDC-tool)	significant effort especially if functional reset is used
check for reconvergence	**low … high (CDC-tool)**	**might be a nightmare**
(…)	(…)	(…)

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

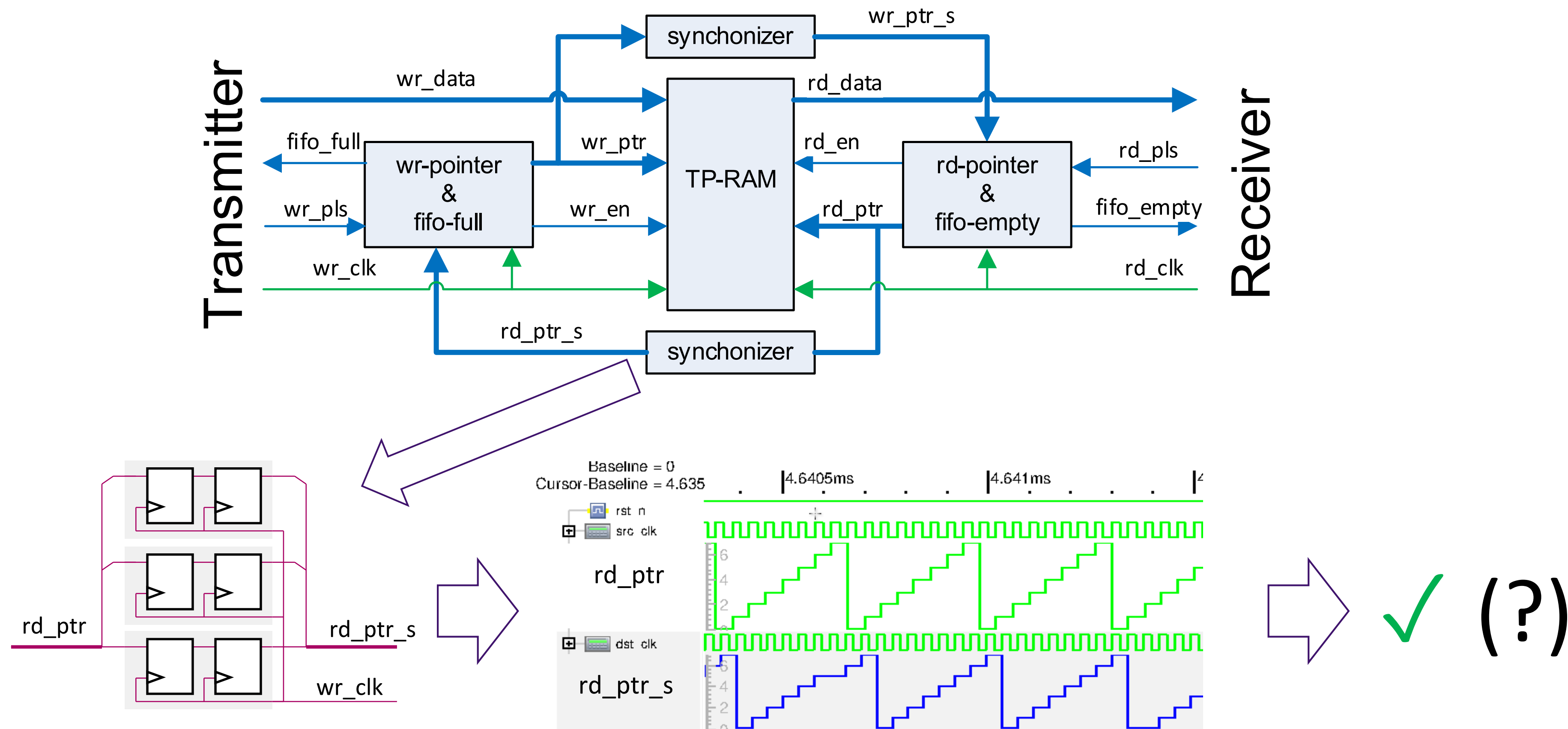
CDC-Reconvergence (example)



challenge: CDC-verification engineer should not the RTL designer (4 eyes)

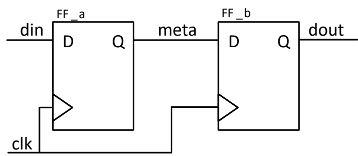
- which synchronizers are exclusive?
- how to ensure that there is no functional glitch (hidden corner case!?)
- if functional glitch is allowed; how to ensure that it does not harm?

simple simulation example: asynchronous FIFO

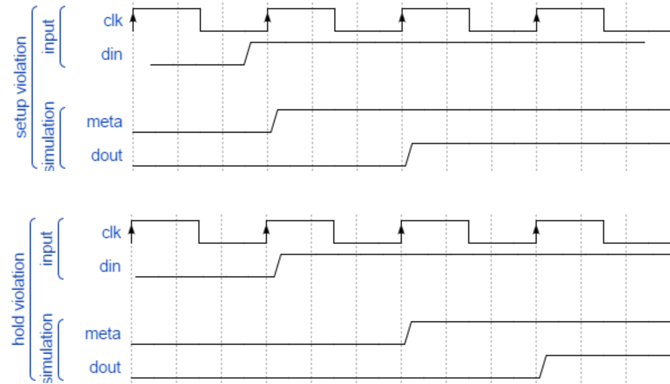


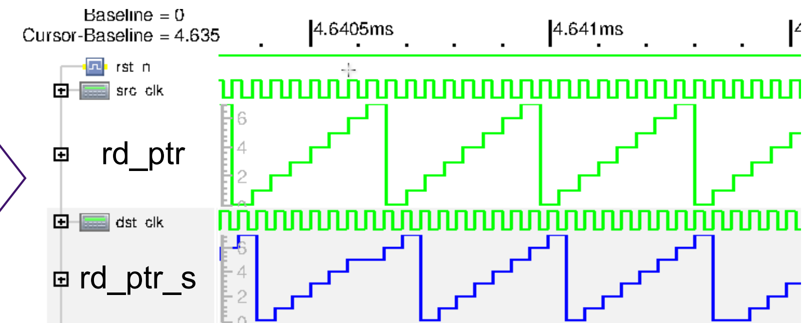
accellera SYSTEMS INITIATIVE
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

simulation <-> silicon discrepancy

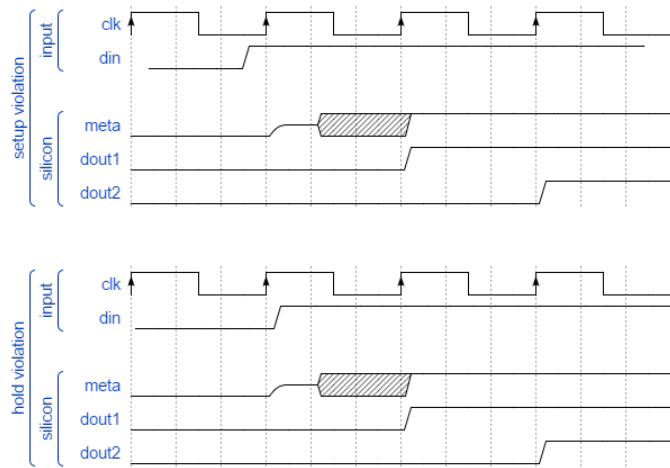


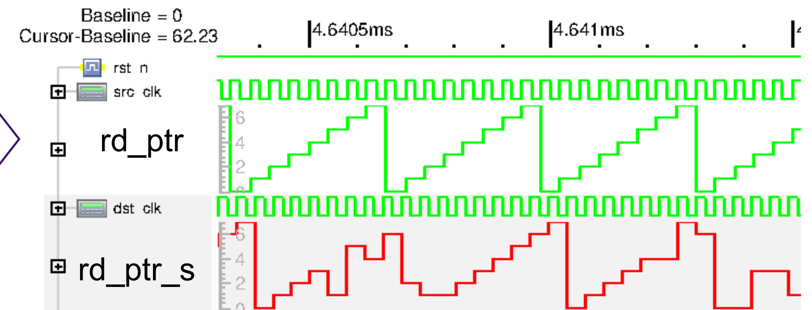
RTL simulation

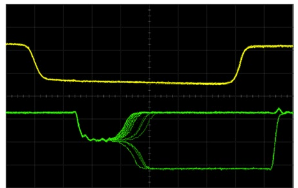




silicon behavior

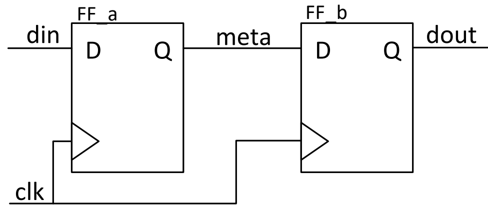




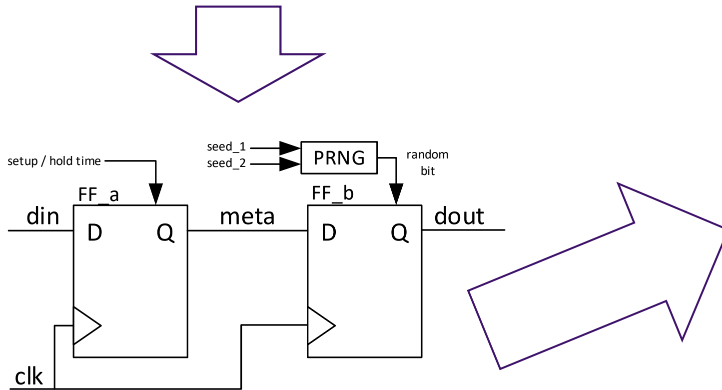


metastability measurement

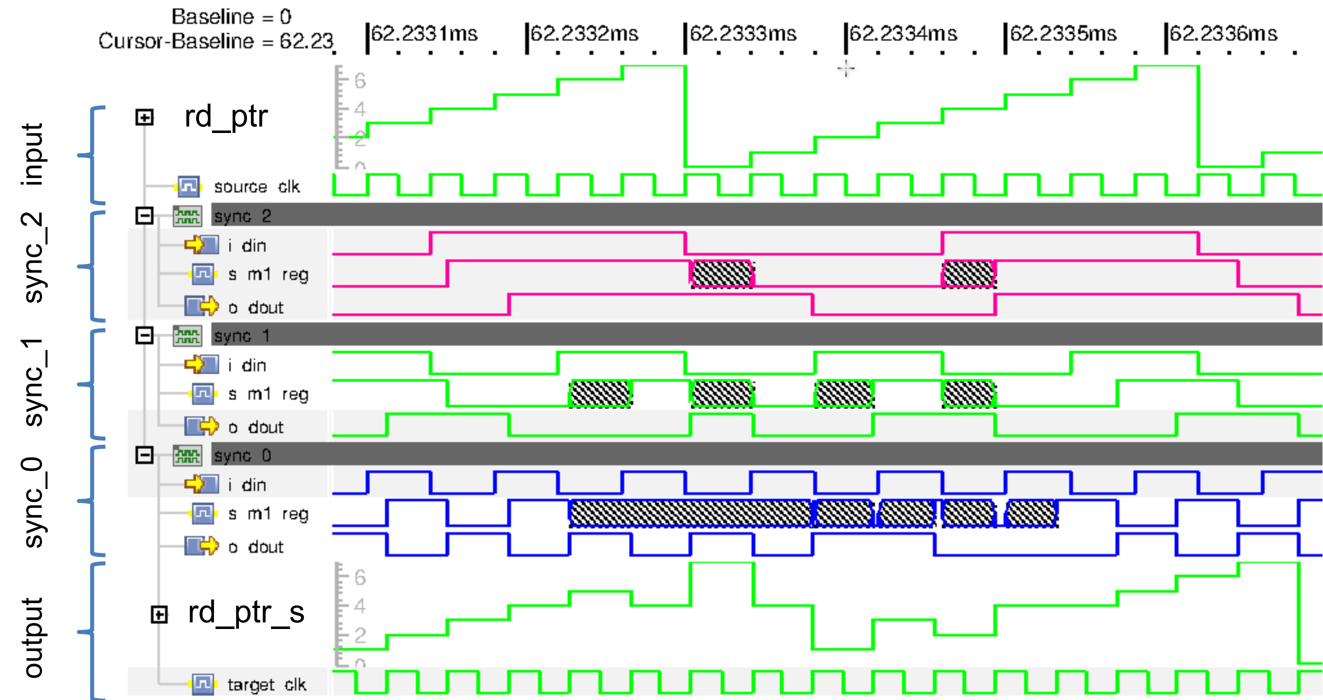
Modelling CDC-Jitter



normal synchronizer;
used for backend / formal checks



jitter-modeling synchronizer;
simulation only!



input and output of 3-bit parallel synchronizer (e.g. FIFO read pointer) and nets of the 3 individual synchronizers (sync_[2:0])

© Accellera Systems Initiative

Randomness of CDC-Jitter generation

Requirements:

- all synchronizers must resolve independent from each other
- same test with different synchronizer behavior must be possible
- any simulation must be reproducible (random stability)

Solution:

- using 2 seeds for PRNG:
  - seed_1: individual on each synchronizer instance
  - seed_2: UVM test seed
- VHDL IEEE.MATH_REAL library provides a function:

  ```
  UNIFORM(SEED1,SEED2:inout POSITIVE; X:out REAL);
  ```

  which matches perfect …

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

generation of device individual seed (seed_1)

- VHDL package containing the synchronizers provides a procedure:

```
100.  SHARED VARIABLE v_nonce     : positive := 1;     -- seed1
101.  PROCEDURE pr_get_nonce (VARIABLE uv_nonce : OUT positive) IS
102.    BEGIN
103.      v_nonce  := v_nonce + 1;
104.      uv_nonce := v_nonce;
105.    END PROCEDURE pr_get_nonce;
```

- procedure is called once by each synchronizer instance
- an individual number is provided each time

© Accellera Systems Initiative

constrained-random-simulation seed (seed_2)

- have a “set” and “get” function in VHDL package

```
200.    SHARED VARIABLE v_uvm_seed  : positive := 6473802;    -- seed2
201.    PROCEDURE pr_get_uvm_seed (VARIABLE uvm_seed : OUT positive) IS
202.    BEGIN
203.      uvm_seed := v_uvm_seed;
204.    END PROCEDURE pr_get_uvm_seed;
205.    PROCEDURE pr_set_uvm_seed (VARIABLE uvm_seed : IN positive) IS
206.    BEGIN
207.      v_uvm_seed := uvm_seed;
208.    END PROCEDURE pr_set_uvm_seed;
```

- procedure `pr_get_uvm_seed` is called once by each synchronizer instance
- the constrained-random-simulation seed is provided

© Accellera Systems Initiative

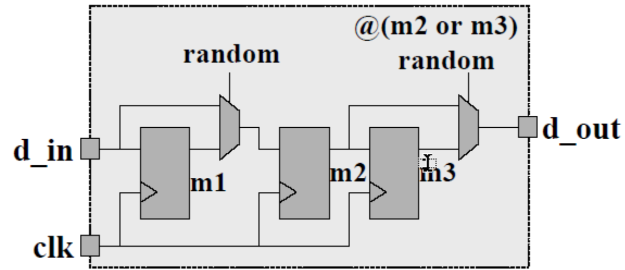
passing constrained-random-simulation seed

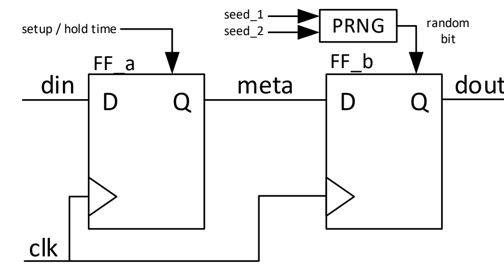
- "set" function is called once at simulation start / after evaluation

```
200.  vhdl procedure 'pr_set_uvm_seed' using interface="(uvm_seed:positive)",
201.     library="worklib", package="cdc_components_pack";
202.
203.  set_cdc_seed() @sys.any is {
204.      'worklib.cdc_components_pack.pr_set_uvm_seed'(covers.get_seed());
205.  };
206.  run() is also {
207.      start set_cdc_seed();
208.  };
```


© Accellera Systems Initiative

comparison with existing solutions





muxing randomly FF-outputs (2006)	resolving randomly meta state (new sheme)
- configuration not needed nor possible	- configuration mandatory and possible
✓ synthesizable description possible	✘ 2nd configuration needed for backend flow
- no differentiation setup/hold (pessimistic)	✓ differentiation setup / hold → more accurate
✘ may fails if clock-gating is used	✓ clock-gating capable
✘ might leads to discussions if synchronizer fails …	✓ metastable net observable in simulation
✓ good for designs without clock-gating and relaxed synchronization constraints (fire and forget)	✓ good for custom synchronization circuits and clock-gated clocks

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

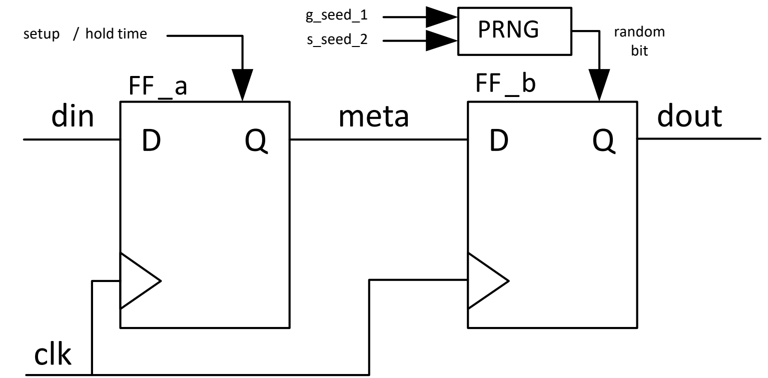
Results of using Jitter-Modeling

- time to fix reconvergence-problems are reduced (to zero)
- does not discourage from using CDC-checking tools
- nice potential having a dedicated module for synchronizer
  - constraints for backend possible:
    - place synchronizer Flip-Flops together
    - protect net between Flip-Flops (e.g. for scan-test insertion)
  - easy to detect wild synchronizers with CDC-checking tool
- mandatory for our projects

© Accellera Systems Initiative

Questions

VHDL Entity




```
ENTITY bst_cdcff IS
  GENERIC( g_rst_val   : std_ulogic := '0';
           g_scan_en   : std_ulogic := '1';
           g_susc_time : natural    := 40000 );

  PORT(    i_din       : IN  std_ulogic;
           i_clk       : IN  std_ulogic;
           i_rst_n     : IN  std_ulogic;
           o_dout      : OUT std_ulogic );
END ENTITY
```

choose reset value
(reset sync. to 1 or 0)

choose if synchronizer
should be added to scan-chain
used by backend tool (e.g. exclude reset sync.)

g_susc_time: setup and hold time
should be slightly smaller than half a clock cycle

© Accellera Systems Initiative

Implementation

```
BEGIN  -- ARCHITECTURE rtl

    s_susc_time <= g_susc_time * 1 ps; -- translate g_susc_time into time

    p_one: PROCESS ( ALL ) IS   -- purpose: 1st stage detect setup / hold;
      VARIABLE clk_time : time := 0 ns;
      VARIABLE din_time : time := 0 ns;
    BEGIN  -- PROCESS p_violate
      IF i_rst_n = '0' THEN
        IF    g_rst_val = '0' AND i_din = '0' THEN s_m1_reg <= ZERO;
        ELSIF g_rst_val = '1' AND i_din = '1' THEN s_m1_reg <= ONE;
        ELSE                                       s_m1_reg <= META; END IF;
      ELSIF i_din'event THEN
        din_time := now;
        IF din_time - clk_time < s_susc_time THEN
          s_m1_reg <= META;                       -- hold viol.; goto meta
        END IF;
      ELSIF i_clk'event AND i_clk = '1' THEN      -- rising clock edge
        clk_time := now;
        IF clk_time - din_time < s_susc_time THEN -- setup violation
          s_m1_reg <= META;
        ELSE
          IF i_din = '1' THEN s_m1_reg <=  ONE;
          ELSE                s_m1_reg <=  ZERO; END IF;
        END IF;
      END IF;
    END PROCESS p_one;
```

```
    p_two: PROCESS (i_clk, i_rst_n) IS              -- purpose: 2nd stage ff
      VARIABLE v_seed1, v_seed2 : positive;         -- seed values for PRNG
      VARIABLE v_rng_boot       : boolean := true;  -- reseed from signal
      VARIABLE v_rand_real      : real;             -- random value [0..1]
    BEGIN  -- PROCESS p_one
      IF i_rst_n = '0' THEN                         -- asynchronous reset
        s_dout_reg <= g_rst_val;
      ELSIF i_clk'event AND i_clk = '1' THEN        -- rising clock edge
        IF v_rng_boot THEN
          v_rng_boot := false;
          pr_get_nonce   ( v_seed1 );               -- get inst.individual seed
          pr_get_uvm_seed( v_seed2 );               -- get UVM seed
          v_seed2 := s_seed;
        END IF;
        CASE s_m1_reg IS
          WHEN ZERO =>  s_dout_reg <= '0';
          WHEN ONE  =>  s_dout_reg <= '1';
          WHEN META =>
            uniform(v_seed1, v_seed2, v_rand_real);   -- generate random number
            IF v_rand_real > 0.5 THEN
              s_dout_reg <= NOT s_dout_reg;
            END IF;
        END CASE;
      END IF;
    END PROCESS p_two;
    o_dout <= s_dout_reg;

END ARCHITECTURE rtl;
```