2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Use of CDC-jitter-modeling in clock-domain-crossing-circuits in RTL design phase

Dr. Jan Hayek, Bosch Sensortec GmbH, Munich, Germany (jan.hayek@de.bosch.com)
Jochen Neidhardt, Bosch Sensortec GmbH, Munich Germany (jochen.neidhardt@de.bosch.com)
Robert Richter, Bosch Sensortec GmbH, Munich Germany(robert.richter2@de.bosch.com)

***Abstract*— Checking a design for proper synchronization across different clock domains is usually started in the mid or end of the RTL design phase. Synchronization issues may have significant impact on a project, varying from a delay in the schedule to even another iteration of the design cycle by going back to the concept phase. We introduce a new behavioral description of a 2-Flip-Flop synchronizer (the basic component of most synchronization circuits) that is able to model CDC-jitter within any simulation based verification framework. The technique we used has proven to significantly reduce the risk of identifying new problems late in the project cycle.**

Keywords: CDC, RTL, UVM, design-methodology

I. INTRODUCTION

Typical techniques to ensure the correct function of synchronization circuits are: reviews, reuse of known good circuits, formal assertions that check preconditions and usage of specialized formal or static CDC-tools. Designs with multiple internal and external gated clocks with switchable frequencies and low-latency requirements for data transfer across clock domain boundaries often require the design of new synchronization circuits. Using CDC-jitter-modeling in addition is closing the gap between strict deterministic circuit behavior during simulation and nondeterministic behavior of silicon circuits. If used instantly from the beginning of the design phase, CDC-reconvergence problems show up with very high probability in the first simulations and reduce the time required to fix the issue in later design phases. The key component of the proposed solution is a 2-Flip-Flop synchronizer model that models a metastability resolution employing a pseudo random number generator. The model allows individual behavior of each synchronization circuit in the design and also throughout different simulations with guaranteed reproducibility (random stability).

The advantage over existing solutions of CDC-jitter modeling[1][2] is an improved precision that helps verifying custom synchronizer circuits for low-power design that use extensive clock-gating. These custom synchronizer circuits could not be verified when using the existing CDC-jitter modeling because of too pessimistic behavior and problems with gated clocks as input.

II. METASTABILITY

Passing data between asynchronous clock-domains causes registers to become metastable with a certain probability, which on the other hand causes unpredictable behavior of the subsequent gates. One of the state-of-the-art solutions is to use a 2-Flip-Flop synchronizer to prevent this potential metastability from propagating to functional logic (see Figure 1).

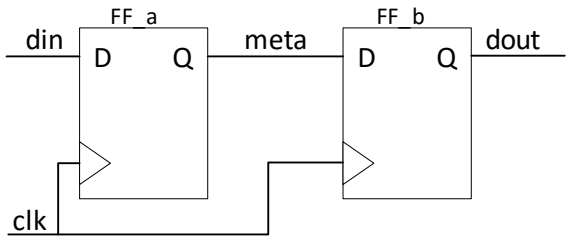

Figure 1 2-Flip-Flop synchronizer. Data on signal **din** is synchronized to clock domain using **clk**; metastability caused by setup/hold violation of **FF_a** should be resolved within one clock-cycle. Valid data is then sampled by **FF_b** to generate the synchronized signal **dout**.

Digital RTL simulations do not incorporate the concept of metastability and will always behave in the same deterministic way. Resolving metastability in real silicon is not deterministic. If a register becomes metastable, it resolves with a high probability[5] within a clock-cycle to either high or low with an unpredictable probability called CDC-jitter (see Figure 2 and Figure 3 for resolving from setup- and hold-violation in simulation and real circuit).

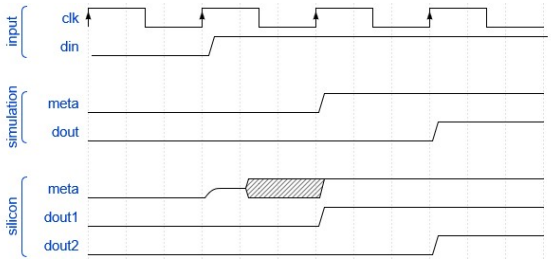

Figure 2: Resolving from metastability caused by hold conflict: **dout** may arrive one clock-cycle earlier than simulated. Hatched area: signal has resolved to logic one resulting in **dout1** or logic zero resulting in **dout2**.

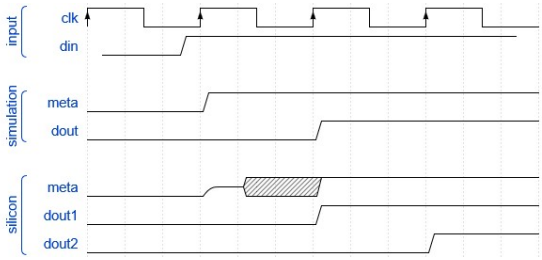

Figure 3: Resolving from metastability caused by setup conflict: **dout** may arrive one clock-cycle later than simulated. Hatched area: signal has resolved to logic one resulting in **dout1** or logic zero resulting in **dout2**.

III. RECONVERGENCE

CDC-jitter as described in chapter II is not a problem by itself, especially if the designer is aware of the random delay and does not expect the signal to be synchronized at a specific time. However, synchronizing multiple related signals with synchronizers can lead to reconvergence problems due to CDC-jitter, since each synchronizer delays signals in a non-predictable way. It is hard to track these problems, because reconvergence issues are not limited to the logic cone(s) directly after the synchronizers, but can also encompass multiple subsequent register stages. Due to the deterministic behavior of a synchronizer in RTL-simulation, these reconvergence problems will not show up in simulation and dedicated tools need to be utilized to search for such issues.

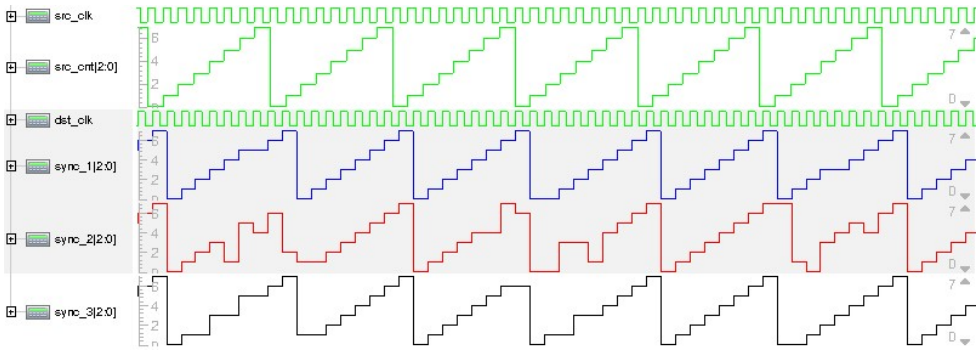

Figure 4: Synchronization of a 3-bit counter **src_cnt** (green) from **src_clk** domain to **dst_clk** domain: **sync_1** (blue) using one 2FF sync per bus-signal (typical simulation result); **sync_2** (red) using same synchronization as **sync_1** but with CDC-jitter emulation; **sync_3** (black): doing gray-encoding before synchronizing; also simulated with CDC-jitter.

An example for insufficient simulation behavior is shown in Figure 4: A counter value **src_cnt** (green) is synchronized from domain **src_clk** into domain **dst_clk**. This could be for example a read- or write-pointer of an asynchronous FIFO. Synchronizing each bit to solve metastability appear to work well in simulation (see: **sync_1** (blue)). However, as metastability is resolved randomly in real silicon (see: **sync_2** (red)), corrupt values will be generated that disrupt the design but do not show up in simulation (neither at RTL nor gate level). One design solution for this case would be to synchronize the pointers in gray-encoded representation. Using gray-encoded counter values will ensure consistency between read- and write-pointer across clock domains, even with random CDC-jitter (see: **sync_3** for one pointer(black)).

IV. MODELLING CDC-JITTER

The basic concept of the CDC-jitter modelling 2FF synchronizer is shown in Figure 5. The first Flip-Flop (**FF_a**) monitors the time between **din** and **clk** events. If the time between **clk** and **din** events is below a given setup or hold time, **FF_a** will become metastable. The signal **meta** between both Flip-Flops is modeled in a way that it can represent metastability in addition to the normal high and low value (data type for net **meta** is modeled as an enumeration type with values: ONE, ZERO and META). In this case the second Flip-Flop (**FF_b**) will resolve to a random value delivered by the pseudo-random-number-generator (**PRNG**) at the next clk edge.

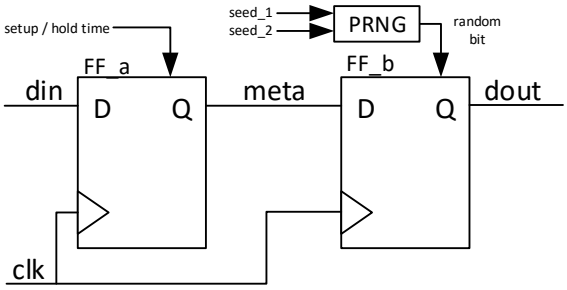

Figure 5: Basic structure of the CDC-jitter modelling 2FF synchronizer.

The PRNG is built using the uniform() procedure which is part of the VHDL ieee.math_real package. It is started with 2 seeds (**seed_1** and **seed_2**). One seed is set up to be unique for each instance (see chapter VI on page 4), the other one uses the seed for constrained-random-simulation. Using this approach also makes the effect of metastability visible in simulations, which helps debugging a circuit (see Figure 6 as an example).

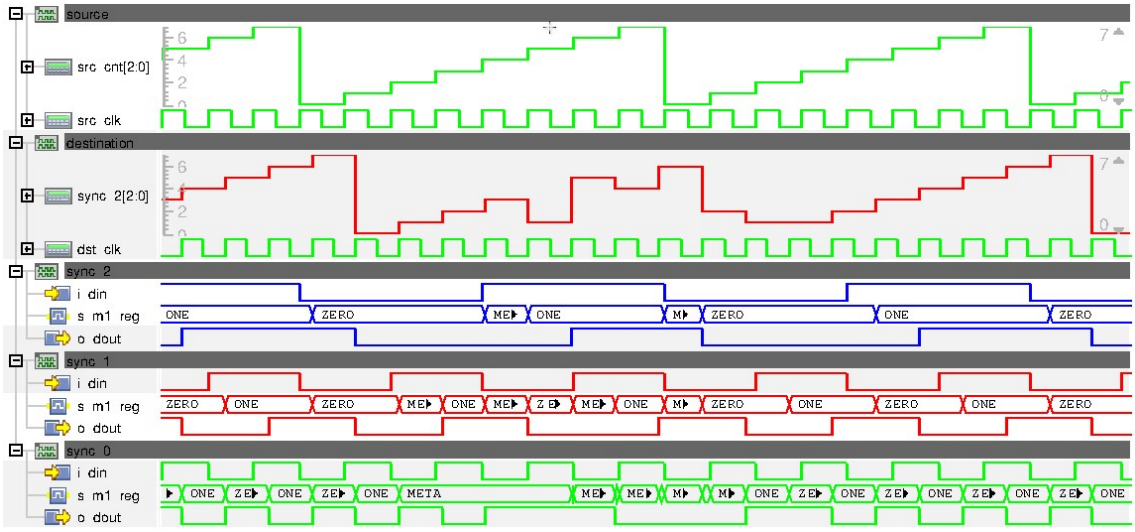

Figure 6 net-states of all three synchronizers when synchronizing a counter.

Figure 6: Shows input, internal net and output of each synchronizer when synchronizing each bit of a three bit counter individually.

V. COMPARISON WITH EXISTING APPROACHES

CDC-jitter modelling in simulation has already been introduced in 2006 [1][2]. The models in these solutions use one or two multiplexers to delay the signal by one clock-cycle or to bypass Flip-Flops (see Figure 7).

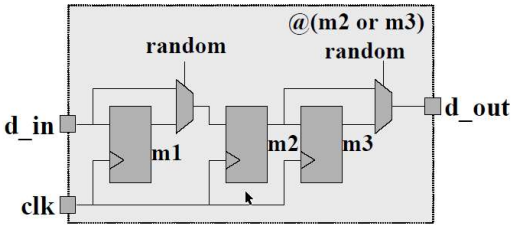

Figure 7 CDC-jitter modelling Synchronizer described in [2]

The advantage of this solution are the possibility to describe the model in a synthesizable way (just exclude one Flip-Flop and multiplexer by synthesis-off pragmas) and the easy configuration (no setting of susceptibility-window needed). One disadvantage of this circuit is its misbehavior in an environment that makes usage of clock-gating (functional clock-gating or clock-gating for power reduction).

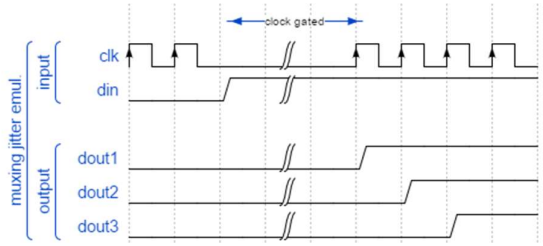

Figure 8: Three possible outcomes in clock-gating environment if jitter-modeling with muxing[2] is used (Figure 7)

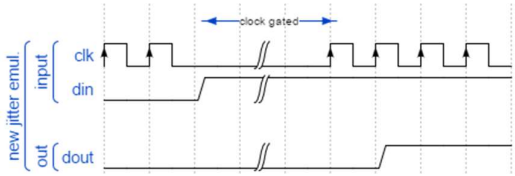

Figure 9: Outcome in clock-gating environment with the proposed jitter-modeling (Figure 5)

The muxing jitter-modeling[2] does not distinguish between setup-/hold-violation or no violation at all. If a steady clock is present at the synchronizer, this works fine. The results however might be more pessimistic than needed as a potential hold violation is treated the same way as a setup violation on the next clock edge, which does not match the real possible outcomes. If the clock is gated, the outcome may break protocols giving a false negative result. Figure 8 shows the three possible results of the muxing synchronizer[2]. A change of **din** is considered as a possible hold-violation at the last clock-edge before the clock is gated and a possible setup-violation of first clock-edge after the clock is released. In fact, there is no possible violation at all and there is only one possible outcome also shown by the new jitter-modeling synchronizer. In contrast to our solution, using the muxing jitter-modeling will not annotate the occurrence of metastability which might cause confusion during simulation reviews.

VI. PRNG SEED-DISTRIBUTION

Special care needs to be taken to generate the seed for the PRNG of each synchronizer (see Figure 5 on page 3) as the same seed will cause the same stream of random numbers which causes same behavior witch is not desirable. Furthermore, reproducibility (random stability) of a simulation must be ensured. Independent behavior of all PRNGs can be archived by assigning a unique number only used once (Nonce) to each instantiation of a synchronizer and using this number as a first seed.

This is done by defining the procedure **pr_get_nonce** (see line 115 of Listing 1) in the package also containing the cdc-jitter model itself. That procedure provides a new, unique value each time it is called. This procedure is called exactly once by each cdc-jitter model instantiation upon a first clock or reset event initializing the individual seed of the instance. (see Chapter VIII The Architecture on Page 7 for further details). A second procedure **pr_set_uvm_seed** is defined to pass the UVM seed to the simulation. This procedure is called once after evaluation at first simulation event (see line 204 of Listing 2). The UVM seed is stored in a variable **v_uvm_seed** and is fetched by each cdc-jitter model instantiation by calling the procedure **pr_get_uvm_seed** at the same time as **pr_get_nonce**.

```
100.  PACKAGE cdc_components_pack IS
101.    -- pragma synthesis_off
102.    SHARED VARIABLE v_nonce     : positive := 1;          -- seed1
103.    SHARED VARIABLE v_uvm_seed  : positive := 6473802;    -- seed2
104.    PROCEDURE pr_get_nonce    (VARIABLE uvm_nonce : OUT positive);
105.    PROCEDURE pr_get_uvm_seed (VARIABLE uvm_seed  : OUT positive);
106.    PROCEDURE pr_set_uvm_seed (VARIABLE uvm_seed  : IN  positive);
107.    -- pragma synthesis_on
108.  COMPONENT bst_cdcff
109.       […]
110.  END COMPONENT;
111.  END cdc_components_pack;
112.
113.  PACKAGE BODY cdc_components_pack IS
114.    --pragma synthesis_off
115.    PROCEDURE pr_get_nonce (VARIABLE uvm_nonce : OUT positive) IS
116.    BEGIN
117.      IF v_nonce = 1 THEN
118.        ASSERT v_uvm_seed /= 6473802
119.        REPORT "unknown uvm-seed (call pr_set_uvm_seed from tb!)" SEVERITY warning;
120.      END IF;
121.      v_nonce   := v_nonce + 1;
122.      uvm_nonce := v_nonce;
123.    END PROCEDURE pr_get_nonce;
124.
125.    PROCEDURE pr_get_uvm_seed (VARIABLE uvm_seed : OUT positive) IS
126.    BEGIN
127.      uvm_seed := v_uvm_seed;
128.    END PROCEDURE pr_get_uvm_seed;
129.
130.    PROCEDURE pr_set_uvm_seed (VARIABLE uvm_seed : IN positive) IS
131.    BEGIN
132.      v_uvm_seed := uvm_seed;
133.    END PROCEDURE pr_set_uvm_seed;
134.    -- pragma synthesis_on
135.  END cdc_components_pack;
```

Listing 1: Declaration and implementation of needed support procedures pr_get_nonce, pr_get_uvm_seed and pr_get_uvm_seed to pass seeds to ensure independent behavior of all synchronizers

```
200.  extend bst_cdc_lib_tb_env_u {
201.    vhdl procedure 'pr_set_uvm_seed' using interface="(uvm_seed:positive)",
202.    library="worklib", package="cdc_components_pack";
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.    };
209.  };
```

Listing 2: specman e testbench environment snipped that passes the UVM-seed to simulation after evaluation at first simulation event

VII. CONFIGURATIONS AND INTERFACE

At least two configurations for the 2 flip-flop synchronizer are needed in the flow:

- Jitter-modeling configuration: used in RTL simulation / regression only. This configuration is not synthesizable and needs to be exchanged with an implementation configuration in the physical implementation flow
- Implementation configuration: synthesizable description of the 2-Flip-Flop synchronizer

Both configurations need to use the same interface to support the replacement. The interface of the 2-Flip-Flop synchronizer is shown in Listing 3. The following definitions apply:

A. *Generics:*

- g_rst_val: defines the reset value of the synchronizer (applies to both configurations)
- g_scan_en: scan enable: is ignored in both configurations but picked up by scan-insertion step in the design backend. This parameter allows the designer to explicit exclude a synchronizer from the scan chain, e.g. a reset synchronizer.
- g_susc_time: susceptibility time: defines setup and hold time before and after the clock edge in which metastability will be generated in the first Flip-Flop. Recommended setting is "a little less than half a clock cycle" to let the model clearly distinguish between setup and hold violation which have different outcomes. It is not recommended to enter real Flip-Flop timings here. Higher values will increase the occurrence of metastability in simulations (compared to real behavior) and thus ensures valuable results in short regressions.

B. *Ports:*

- i_din: data input
- i_clk: clock of target clock domain
- i_rst_n: reset, active low; reset value is determined by generic parameter g_rst_val.
- o_dout: i_din synchronized to i_clk clock domain

```
300.  ENTITY bst_CDCff IS
301.    GENERIC( g_rst_val   : std_ulogic := '0'; -- reset value
302.             g_scan_en   : std_ulogic := '1';
303.             g_susc_time : natural    := 40000 );  -- susceptibility window
304.    PORT(    i_din       : IN  std_ulogic;
305.             i_clk       : IN  std_ulogic;
306.             i_rst_n     : IN  std_ulogic;
307.             o_dout      : OUT std_ulogic );
308.  END ENTITY bst_CDCff;};
```

Listing 3: Interface description of configurable 2-Flip-Flop synchronizer

VIII. THE ARCHITECTURE

Listing 4 shows the architecture of the jitter-modeling synchronizer. The synchronizer is part of the cdc-components package (see Listing 1) that also contains the essential functions for proper initialization of the PRNG contained in each synchronizer (see Chapter VI on Page 4).

PRNG initialization (Line 435-436) is triggered once after process **p_two** (Line 425-449) is triggered the first time by either a clock or reset event.

```
400.  BEGIN  -- ARCHITECTURE rtl
401.    s_susc_time <= g_susc_time * 1 ps; -- translate g_susc_time into time
402.    p_one: PROCESS ( ALL ) IS          -- purpose: 1st stage ff; detect setup and hold;
403.      VARIABLE clk_time : time := 0 ns;
404.      VARIABLE din_time : time := 0 ns;
405.    BEGIN  -- PROCESS p_violate
406.      IF i_rst_n = '0' THEN
407.        IF    g_rst_val = '0' AND i_din = '0' THEN s_m1_reg <= ZERO;
408.        ELSIF g_rst_val = '1' AND i_din = '1' THEN s_m1_reg <= ONE;
409.        ELSE                                      s_m1_reg <= META; END IF;
410.      ELSIF i_din'event THEN
411.        din_time := now;
412.        IF din_time - clk_time < s_susc_time THEN
413.          s_m1_reg <= META;                       -- hold violation; goto meta
414.        END IF;
415.      ELSIF i_clk'event AND i_clk = '1' THEN      -- rising clock edge
416.        clk_time := now;
417.        IF clk_time - din_time < s_susc_time THEN  -- setup violation
418.          s_m1_reg <= META;
419.        ELSE
420.          IF i_din = '1' THEN s_m1_reg <=  ONE;
421.          ELSE                s_m1_reg <=  ZERO; END IF;
422.        END IF;
423.      END IF;
424.    END PROCESS p_one;
425.    p_two: PROCESS (i_clk, i_rst_n) IS              -- purpose: 2nd stage ff
426.      VARIABLE v_seed1, v_seed2 : positive;         -- seed values for random generator
427.      VARIABLE v_rng_boot       : boolean  := true; -- reseed from signal
428.      VARIABLE v_rand_real      : real;             -- random real value in range 0 to 1.0
429.    BEGIN  -- PROCESS p_one
430.      IF i_rst_n = '0' THEN                         -- asynchronous reset (active low)
431.        s_dout_reg <= g_rst_val;
432.      ELSIF i_clk'event AND i_clk = '1' THEN        -- rising clock edge
433.        IF v_rng_boot THEN
434.          v_rng_boot := false;
435.          pr_get_nonce   ( v_seed1 );               -- get instance individual seed
436.          pr_get_uvm_seed( v_seed2 );               -- get simulation seed
437.          v_seed2 := s_seed;
438.        END IF;
439.        CASE s_m1_reg IS
440.          WHEN ZERO =>  s_dout_reg <= '0';
441.          WHEN ONE  =>  s_dout_reg <= '1';
442.          WHEN META =>
443.            uniform(v_seed1, v_seed2, v_rand_real);   -- generate random number
444.            IF v_rand_real > 0.5 THEN
445.              s_dout_reg <= NOT s_dout_reg;
446.            END IF;
447.        END CASE;
448.      END IF;
449.    END PROCESS p_two;
450.
451.    o_dout <= s_dout_reg;
452.
453.  END ARCHITECTURE rtl;
```

Listing 4: VHDL architecture of the CDC-jitter-modeling 2-Flip-Flop synchronizer

IX. RESULTS AND CONCLUSION

Based on our experience, it is not recommended to integrate the CDC-jitter modelling synchronizer into an existing project. Due to the random shift of timings at the model boundary, there is a high probability that UVM monitors or self-checking test-benches will fail and need to be adjusted to tolerate that behavior. CDC-checking tools provide a much less intrusive solution to search for reconvergence issues at this stage of the project. If, however, used from project start, the RTL designers will spot CDC-reconvergence-problems instantly after designing a block, even if no CDC-checking tool is in place. Early spotted reconvergence-problems can be fixed instantly, which clearly needs much less time than problems spotted during CDC-checking as a post-design effort. Much time is needed to understand the finding, fix it, review it and / or change the design concept if design constraints cannot be met anymore. Depending on design constraints there might be better solutions for jitter-emulation models than the proposed solution. A clear advantage of the proposed solution is the ability to work well in a clock-gated environment. Using a muxing jitter-emulation model proposed in[2] might be the better solution for systems with steady clock as it needs less configuration and can be described in a synthesizable way.

Having a dedicated module for the 2-Flip-Flop-synchronizer has some additional advantages besides jitter-emulation. It allows easy constraining on module level to ensure that the two Flip-Flops are placed together in place-and-route and that the net between the two Flip-Flops is exempt from automatic optimization by (both are essential to minimize and guarantee Mean Time Between Failure (MTBF) of the synchronizer).

Jitter-modeling does not disengage from using dedicated CDC-checking tools:

- It does not check for the synchronizer inputs to be free of glitches
- It does not check for proper synchronization of design inputs
- Design may still have convergence-problems not hit by chance in constrained-random simulation

However, the effort to setup a design with CDC-jitter-modeling compared to the benefit gained made this technique mandatory for our projects.

REFERENCES

[1] Hin-Kwai Lee, Methodology for verifying multi-cycle and clock-domain-crossing logic using random flip-flop delays, US Patent 7,289,946

[2] Mark Litterick, Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using System Verilog Assertions, DVCon 2006

[3] IEEE Std 1076.2 https://standards.ieee.org/downloads/1076/1076.2-1996/math_real.vhdl

[4] K. Meade, S. Rosenberg, A Practical Guide to Adopting the Universal Verification Methodology, 2nd ed, Cadence Design Systems 2013.

[5] Ang Boon Chong, Product Level MTBF Calculation, 2014 Fifth International Conference on Intelligent Systems, Modelling and Simulation