Verification strategies and modelling for the uninvited guest in the system: Clock Jitter

Deepak Nagaria, Synopsys India Pvt. Ltd., Delhi, India (*dnagari@synopsys.com*)

Vikas Makhija, Synopsys India Pvt. Ltd., Noida, India (*vikasm@synopsys.com*)

Apoorva Mathur, Synopsys India Pvt. Ltd., Noida, India (*apoorvam@synopsys.com*)

***Abstract*— This paper presents the modelling of various clock generators using System Verilog to mimic the behavior of various types of jitter in the simulation clock to validate that the design is efficiently able to cope with the allowable jitter in the clock signal. The motivation for this work comes during development of LPDDR verification environment which is a both edge (rising and falling) triggered system and a little jitter (permissible) in the clock can cause timing as well as data sampling issues which are not expected. This paper also briefly describes the different types and characteristics of jitter.**

Keywords*— jitter; periodic jitter; duty cycle jitter; random jitter; deterministic jitter; clock generator; ppm; data eye width; bit errors

I. INTRODUCTION

With the advancement of technology, designs are getting more and more complex. With SOCs, NOCs, multi core processors and the ever-increasing clock speed, it has been a challenge to keep up with the requirement of high performance. Jitter plays a vital role in degrading the overall performance of a system for both serial bus and parallel bus architecture, and proves even more significant for high clock frequencies, by reducing the overall system performance. It has become very much important for design and verification engineers to ensure that the design is full proof to handle the allowable jitter such as maintaining the data integrity before fabricating the design on real silicon. To achieve this a verification environment that can model the behavior of a typical jitter in the clock signal and test the design at all the boundary conditions of the jitter is needed at simulation level.

The work was motivated during the development and engagement phase of LPDDR Verification IP. LPDDR is a low power dual data rate DRAM which operates at both rising edge and falling edge of the clock signal. There are several timing constraints imposed by the protocol which needs to be satisfied for the proper functioning. However, a little jitter in a single clock can cause timing violations while it should have been accepted without any violation until the jitter is within permissible limits and that motivated us in modeling the clock generation in such a way that it will create the real-time scenarios and make the verification of the design more robust.

II. JITTER CHARACTERISTICS AND TYPES OF JITTER

Jitter is the short-term deviation or shift in the edges of a periodic signal from their ideal positions.

With the increase in clock frequency the data eye width gets reduced and as soon as jitter appears in the system the eye width further gets reduced and increases the possibility of bit error. Higher the bit error rate, lower the performance of system.

The optimal position for sampling a data bit is at the center, providing the maximum tolerance for bit sampling error possibility. If the jitter causes a shift in the edges more than half unit interval than there will be under sampling or over sampling of the data bit.

Jitter can be broadly classified in two types: random jitter and deterministic jitter

Random Jitter: It is primarily caused due to thermal variations and collective noise of electrical components in the system. It is unbounded in nature that is why it cannot be predicted however it follows Gaussian distribution.

Deterministic Jitter: It is bounded in nature with a predictable distribution. The sources for deterministic jitter are EMI, crosstalk, inter symbol interference and power supply ripple. Deterministic jitter can be further sub classified into Periodic jitter and Duty cycle distortion.

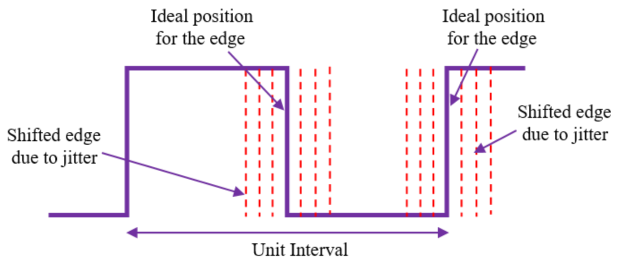
Ideal position for the edge
Ideal position for the edge
Shifted edge due to jitter
Shifted edge due to jitter
Unit Interval

Figure 1. Shift in edges of periodic signal due to jitter

III. Modelling Of Jitter In Clock Generator Module

The bandwidth and nature of total jitter is a cumulative factor of random and deterministic jitter along with noise introduced by the electrical components. The proposed verification environment has the clock generators which can mimic the effect of various possible jitter on the clock signal of the system. In this paper the implemented strategies for verification are by inducing the jitter through modeling of the clock generators. The strategies used are: -

- Random jitter following Gaussian distribution
- Periodic jitter
- Duty cycle distortion jitter
- Total jitter (Random + Deterministic)

The work that has been done as part of this paper is to design the clock generator modules which can follow above mentioned strategies.

A. Random Jitter Following Gaussian Distribution Clock Generator

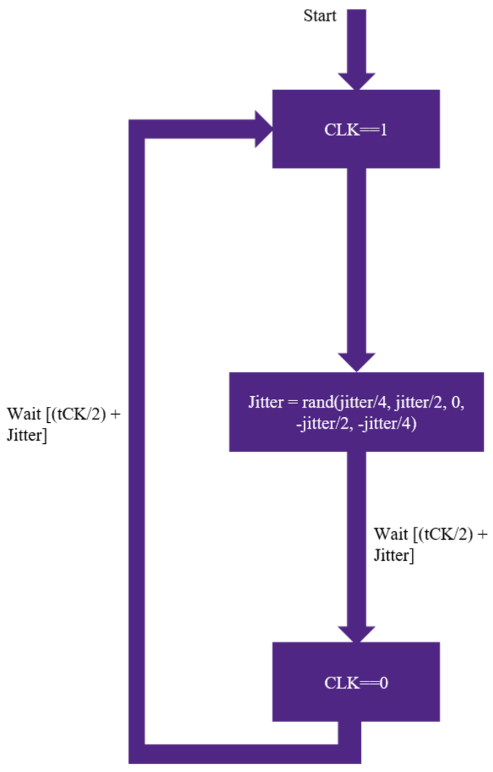

Figure 2. Clock generator with random jitter following Gaussian distribution

This clock generator module has been modeled in such a way that it will follow the Gaussian distribution to induce the behavior of the random jitter in clock signal. The shift in the edges of clock signal is random with constraint values of maximum allowable jitter from 1/2nd, 1/4th, 0, -1/4th and -1/2nd factors with equal weightage in randomization, eventually resulting in an average jitter tending to 0 over the time. The randomization constraints over factors of maximum allowable jitter can have further values such as 1/8th and 1/16th for more granular results.

B. *Periodic Jitter Clock Generator*

This clock generator is designed to induce jitter in a clock cycle periodically. As the Figure 3 depicts the clock period of each cycle is "tCK" except the cycles appearing at Nth location. For every Nth cycle this generator will change the clock period from "tCK" to "tCK + Jitter" and again change it to "tCK" until next Nth cycle to model the behavior of periodic jitter.

This generator can also be used to mimic the PPM (parts per million) attribute in clock signal. PPM gets induced in the clock signal mostly due to the manufacturing inaccuracies of the crystal oscillator. Due to these inaccuracies, the crystal does not generate the clock of precise period i.e. if the crystal is supposed to generate a clock of X ns then it generates a period of either (X+Y) ns or (X-Y) ns where Y <<< X. PPM is additive in nature, as this inaccuracy will be induced in each clock cycle. To generate a PPM induced clock signal, the value of variable N (iteration limit) should be set to 0 and the variable "Jitter" should be replaced with the PPM value that will generate PPM induced clock period.

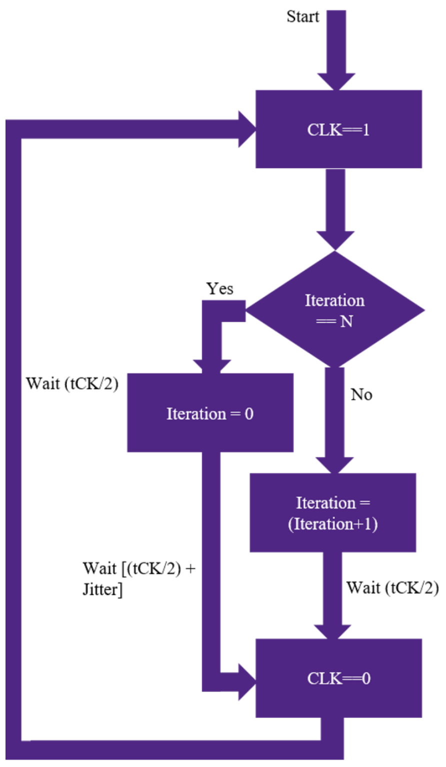

Figure 3. Clock generator with periodic jitter after every N cycles

C. *Duty Cycle Distortion Jitter Clock Generator*

This clock generator shifts the edges (falling edges) of the clock signal such that the overall period of the cycle remains "tCK" however the ON and OFF period of the cycle can be configured other than 50% to model the duty cycle distortion jitter. This type of generator will be very useful in catching timing and data issues in the designs which are both positive edge as well as negative edge triggered like DDR, LPDDR. As the Figure 4 depicts that if the duty cycle distortion is configured to less than 50% then the ON period of this cycle will be "(tCK/2) - Jitter" instead of "tCK/2" and the OFF period will be the remaining tCK period i.e. "(tCK/2) + Jitter". Similarly, if duty cycle distortion is configured to more than 50% then the ON period of this cycle will be "(tCK/2) + Jitter" and OFF period will be the remaining tCK period i.e. "(tCK/2) - Jitter".

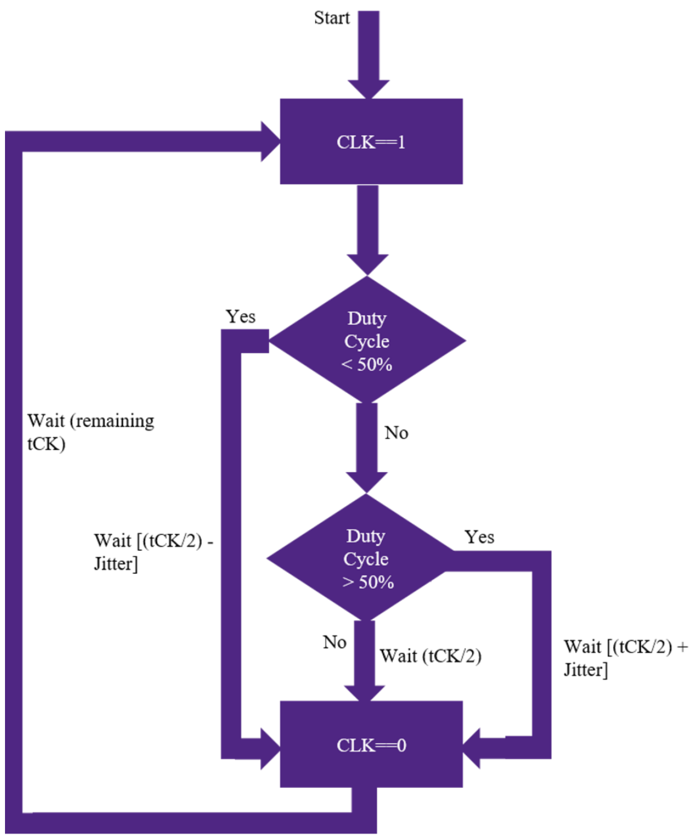

Figure 4. Clock generator with duty cycle distortion jitter

D. *Total Jitter (Random + Deterministic) Clock Generator*

This clock generator models the total jitter which is combination of random jitter and the deterministic jitter. It basically shifts the edges of the clock signal within the minimum and maximum allowable jitter range randomly and does not follow any distribution.

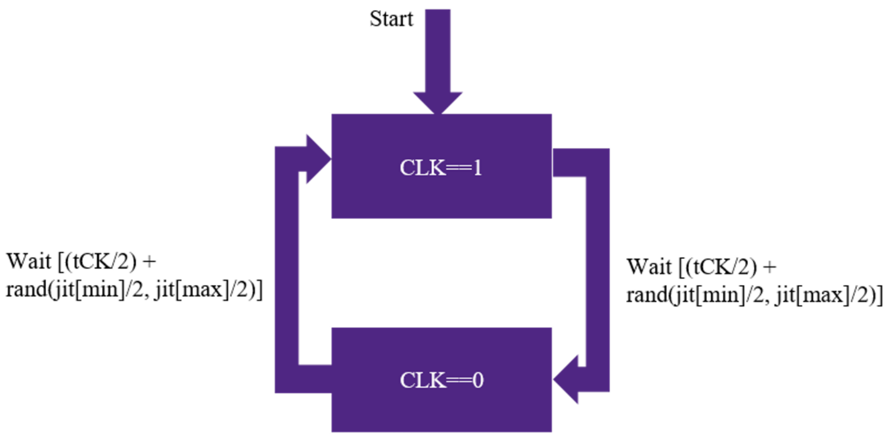
Start
CLK==1
Wait [(tCK/2) + rand(jit[min]/2, jit[max]/2)]
Wait [(tCK/2) + rand(jit[min]/2, jit[max]/2)]
CLK==0

Figure 5. Clock generator with random jitter + deterministic jitter

IV. SYSTEM VERILOG CODE SNIPPET FOR THE GENERATORS

```
task clock_generator(); // Method to generate the clock signal based on ideal clock period or clock period with jitter
  forever begin
    // Checking for any of the clock generator logic with jitter is enable or not
    if ((random_jitter_en == 1'b1) || (periodic_jitter_en == 1'b1) || (duty_cycle_jitter_en == 1'b1) || (total_jitter_en == 1'b1)) begin
       // When any clock generator with jitter is enabled then getting the value of jittered clock period from that generator
      void'(jittery_clk_period_generator());
      #((clock_period_with_jitter/2) * 1ps); // Waiting for jittered clock period before toggling the clock signal
    end
    else begin
      #((clock_period/2) * 1ps); // Waiting for ideal clock period before toggling the clock signal
    end

    CLK = ~CLK; // Toggling the clock signal
  end
endtask
function void jittery_clk_period_generator(); // Method to get the jittered clock period when any jittered clock generator logic is enabled
  if (random_jitter_en == 1'b1 && CLK == 1'b1) begin // Checking for random gaussian distribution jitter generation to be enabled
    void'(this.randomize(index)); // Performing cyclic randomization to get randomized values with equal weightage
    if      (index == 0) clock_period_with_jitter = (clock_period + jitter);   // Setting clock period with jitter value added in it
    else if (index == 1) clock_period_with_jitter = (clock_period - jitter);   // Setting clock period with jitter value removed from it
    else if (index == 2) clock_period_with_jitter = (clock_period + jitter/2); // Setting clock period with jitter/2 vlaue added in it
    else if (index == 3) clock_period_with_jitter = (clock_period - jitter/2); // Setting clock period with jitter/2 value removed from it
    else                 clock_period_with_jitter = (clock_period);            // Setting the ideal clock period
  end
  else if (periodic_jitter_en == 1'b1) begin // Checking for periodic jitter generation to be enabled
    if (CLK == 1'b1) begin // Entering at rising edge of the clock signal
      // Checking when the clock count from preivous jitter injection saturates to the limit N
      // then again inject the jitter in the current clock period
      if (iteration == N) begin
        iteration = 0; // Restting the interation count to increment again from 0 to N
        clock_period_with_jitter = (clock_period + jitter); // Incorporating the periodic jitter value in the current clock period
      end
      else begin
        iteration = iteration + 1; // Incrementing the iteration count
        clock_period_with_jitter = (clock_period); // Setting the ideal clock period as iteration count has not reached the limit
      end
    end
    else begin // At falling edge of the clock setting the ideal clock period for the negative half of clock signal
      clock_period_with_jitter = (clock_period);
    end
  end
  else if (duty_cycle_jitter_en == 1'b1) begin // Checking for duty cycle distortion jitter generation to be enabled
    if (CLK == 1'b1) begin // Checking for the rising edge of the clock to distort the ON period of the clock signal
      if (duty_cycle < 50) begin // Checking for the ON period to be less than 50%
        clock_period_with_jitter = (clock_period - jitter); // Setting clock period with jitter value removed from it
      end
      else if (duty_cycle > 50) begin // Checking for the ON period to be greater than 50%
        clock_period_with_jitter = (clock_period + jitter); // Setting clock period with jitter value added to it
      end
      else begin // Entering when the ON period is exactly 50% i.e. ideal
        clock_period_with_jitter = (clock_period); // Setting the ideal clock period
      end
    end
    // Checking for the falling edge of the clock to set the OFF period for the remaing period of the
    // ideal clock period i.e. (clock_period - clock_period_with_jitter)
    else begin
      if (duty_cycle < 50) begin // Checking for the ON period to be less than 50%
        // Adding jitter value in the clock period as the same was removed during the rising edge i.e. ON period of the cycle
        clock_period_with_jitter = (clock_period + jitter);
      end
      else if (duty_cycle > 50) begin // Checking for the ON period to be greater than 50%
        // Removing jitter value from the clock period as the same was added during the rising edge i.e. ON period of the cycle
        clock_period_with_jitter = (clock_period - jitter);
      end
      else begin // Entering when the ON period is exactly 50% i.e. ideal
        clock_period_with_jitter = (clock_period); // Setting the ideal clock period
      end
    end
  end
  else if (total_jitter_en == 1'b1) begin // Checking for the total jitter generation to be enabled
    // Setting the clock period by adding or removing a jitter value randomly which is within max and min allowed values
    clock_period_with_jitter = (clock_period + (($urandom_range(-jitter, jitter)) % (jitter+1)));
  end
endfunction
```

The implementation of these clock generators is done in such a way that there is no possibility of race condition in the jittered clock signal driving. As the generation or toggling of the clock signal (CLK) is being done in the task "clock_generator" and the calculation of jittered clock period is done in "jittery_clk_period_generator" and the execution of these two methods is sequential and not parallel thus removing the possibility of any kind of race in jittered clock generation. In "clock_generator" method, a forever thread gets opened where it checks whether any kind of jitter injection is enabled or not, if not then CLK signal gets toggled after waiting for "clock_period / 2" time. If any jitter injection is enable, then the method "jittery_clk_period_generator" gets called which populate the variable "clock_period_with_jitter" with the corresponding jitter induced clock period and then the CLK signal gets toggled after waiting for "clock_period_with_jitter / 2" time instead of "clock_period / 2". Since all the processing is done in sequential manner thus removing the possibility of the race conditions in jittered clock signal driving.

V. Experiments and Results

The most common problem in the communication system is to find out the optimum position for sampling of incoming data stream. Ideally when the data eye width is 100% and there are no factors which affects the clock period and data bit pulses then the data can be sampled even at the edges of clock signal itself without incurring any bit errors.

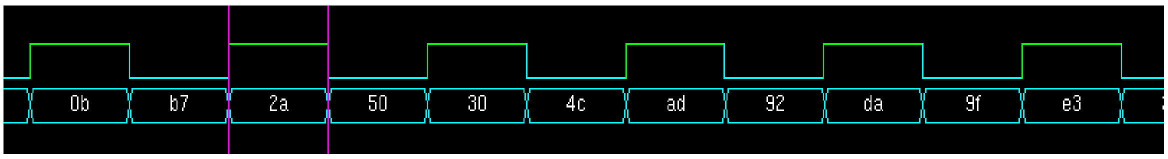

Figure 6. Ideal data bit stream

Now let's talk about the actual scenarios where jitter and PPM takes part in the data communication system. Due to these factors the data eye width gets reduced significantly. Now if the receiver is sampling the bit stream at the point where data is not stable i.e. at the corners of data eye to the induced jitter or PPM period then there will be bit errors as the sampled data information will not be correct at that time.

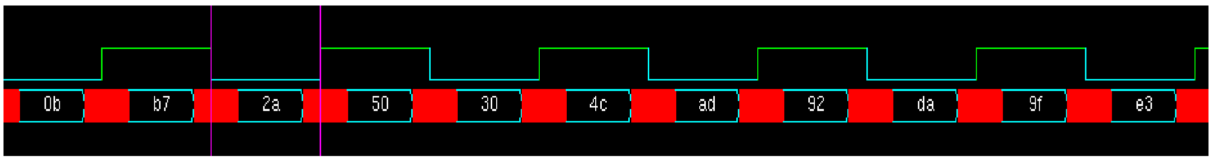

Figure 7. Actual data bit stream

By using the above-mentioned clock generators one can easily control the data eye width and figure out the optimum sampling point as well as the sampling range of the receiver where it can sample the data stream efficiently.

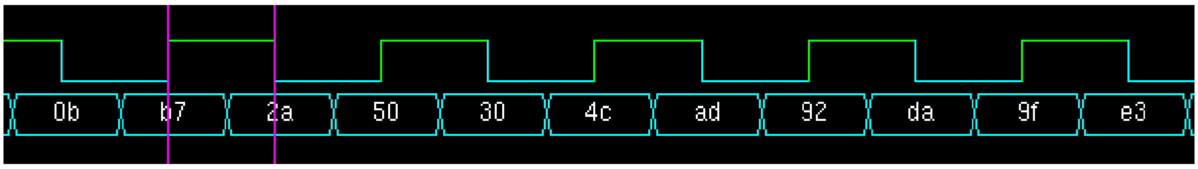

Figure 8. Optimal sampling point at the center of eye

Another problem which generally occurs is usage of hard delays in the behavioral modelling. Hard delays are fine till the clock period is same for each clock cycle as soon as there is even a small jitter appears in the clock period that hard delay won't be able to consider it and eventually will result in timing violations. So, by modelling the jittered clock generators one can easily identify the issues in hard delays for timing consideration.

VI. Summary

The various approaches for modeling jitter are generic in nature and can be associated with any clock generator module to mimic the behavior of jitter for validation of the design. These approaches will ensure the timing and data integrity of the design with allowable jitter in the system by finding the design issues at simulation level.