2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

A Generic Clock UVC for Generating and Testing of High Speed PLL and CDR

Dipanshu, Mukesh Gandhi, Arnab Ghosh & Parag S Lonkar
Samsung Semiconductor India Research (SSIR)
Bangalore, India

***Abstract*- In the ever-evolving landscape of electronics and semiconductor design, clock and data signals are indispensable components. They govern the functionality of countless electronic devices, from smartphones to intricate microprocessors. With the growing complexity of these designs, robust verification processes are essential to ensure their reliability, functionality, and adherence to specifications. This paper introduces a novel approach that seamlessly integrates clock generation, recovery and monitoring, revolutionizing the design verification process. This approach enhances reusability and simplifies integration within Universal Verification Methodology (UVM) Testbenches (TBs). Key components of the approach include the Clock Generator Agent, Clock Checker Agent, and CDR Agent, collectively optimizing clock and data signal verification. The proposed methodology accelerates individual component verification, establishes continuous monitoring of critical parameters during simulations, and promises to expedite verification tasks while enhancing design robustness.**

I. Introduction

As design complexity continues to escalate, rigorous verification processes are paramount to guarantee that the final product meets its intended specifications. However, verifying the correctness and integrity of clock and data signals in these complex designs poses significant challenges, demanding substantial coding efforts and meticulous attention to detail.

While conventional methods often generate clocks within the testbench top, we propose a more streamlined approach using a specialized Clock UVC (Universal Verification Component). A dedicated clock generator agent ensures concise coding practices and facilitates parameter adjustments. It also takes care of the stress testing clock data recovery (CDR) functionality within receiver IPs, which involves injecting jitter and spread spectrum clocking (SSC) to assess robustness. Additionally, a dependable clock checker verifies that the transmitter's PLL clock frequency adhering to permissible ppm deviation (parts per millions) limits. The CDR agent plays an important role of decoding data and extracting the clock signal, ensuring data processing by subsequent verification IPs for protocol level checking. This integration not only accelerates the verification of individual components but also enables continuous monitoring of fundamental parameters throughout simulations.

II. Related Work

Prior research in design verification has primarily focused on individual verification components and separate verification methodologies. Traditional verification approaches often involve extensive manual coding and configuration, leading to time-consuming and error-prone processes. While clock generation and clock-data recovery for verification have been explored in isolation, their seamless integration within a UVC is a novel and distinctive feature of this research.

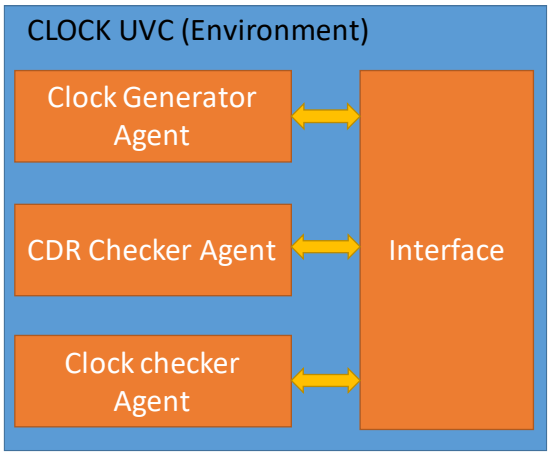

Figure 1: Clock UVC Structure

Central to the proposed approach is a Universal Verification Component (UVC) featuring a three-agent structure: The Clock Generator Agent, the Clock Checker Agent and the CDR Agent.

A. Clock Generator Agent

This agent generates diverse clock signals, accommodating variations in duty cycle, period, frequency, Spread Spectrum Clocking (SSC), jitter, and phase differentials. Additionally, it provides divided clocks and assorted pattern and pulse generators, all equipped with automatic clock checkers. This block is useful in generating the basic clocks and signals that are needed by the design to operate normally. This block is also capable of generating variations in these clock/pulse signals to ensure that the DUT is tested in all corner cases, and the functionality is not changed if there are some noise into these signals. All of these functionalities can be controlled by programming the different config values responsible for them.

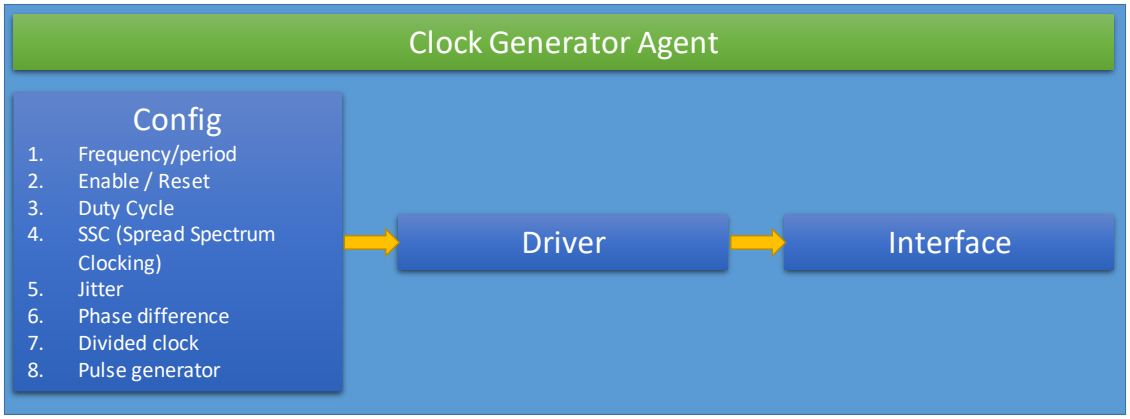

Figure 2 : Clock Generator Agent

B. Clock Checker Agent

This agent meticulously verifies essential clock parameters, including period, frequency, duty cycle, SSC, jitter, phase alterations, clock division factors relative to other clocks, presence of X or Z clocks, and identification of clock glitches. This checker can also extract average clock frequency and other information over multiple clock cycles along with ppm deviation.

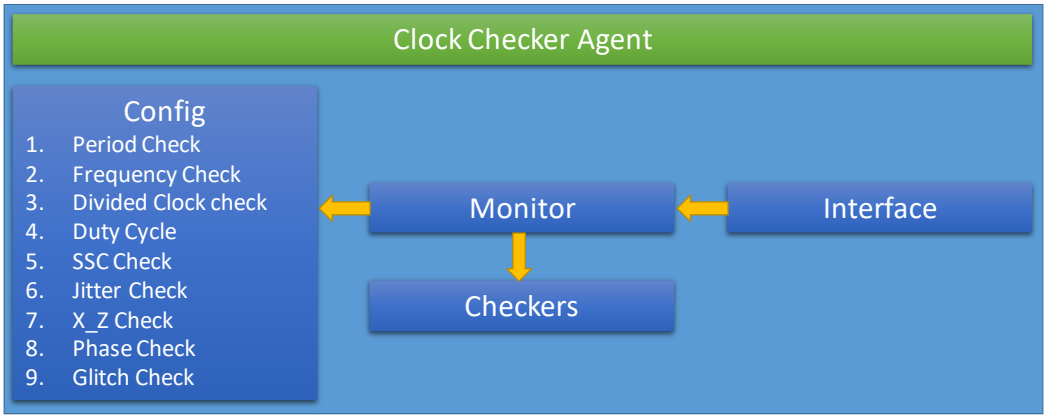

Figure 3 : Clock Checker Agent

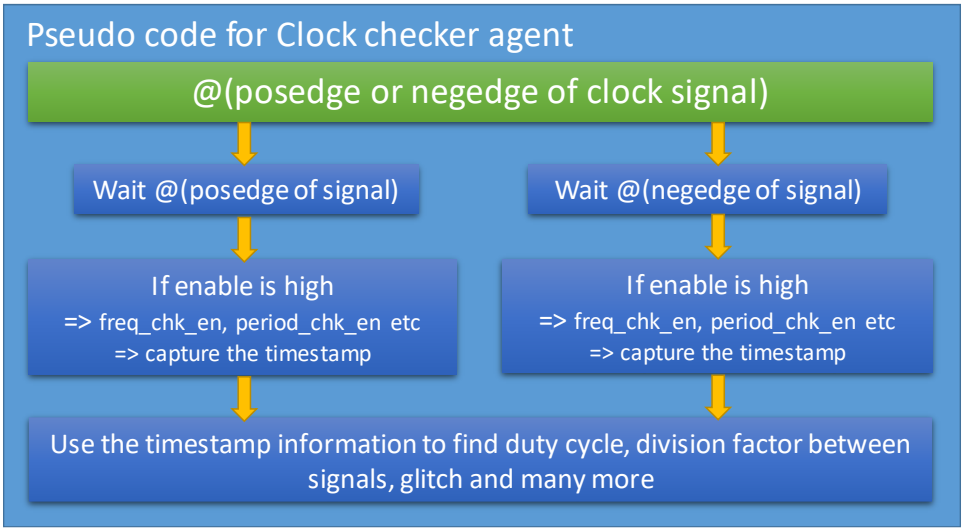

Figure 4 : Pseudo code for clock checker agent

C. CDR Agent

This agent adeptly manages and extracts serialized data from diverse data lines, boasting proficiency in handling noisy components like Jitter, SSC-included variations, phase shifts, and various encoding modes. The CDR Agent ensures precise sampling of noisy data, calculation of UI (Unit Interval) and extraction of clock, which enables seamless subsequent processing of data by Verification IPs for protocol level checking.

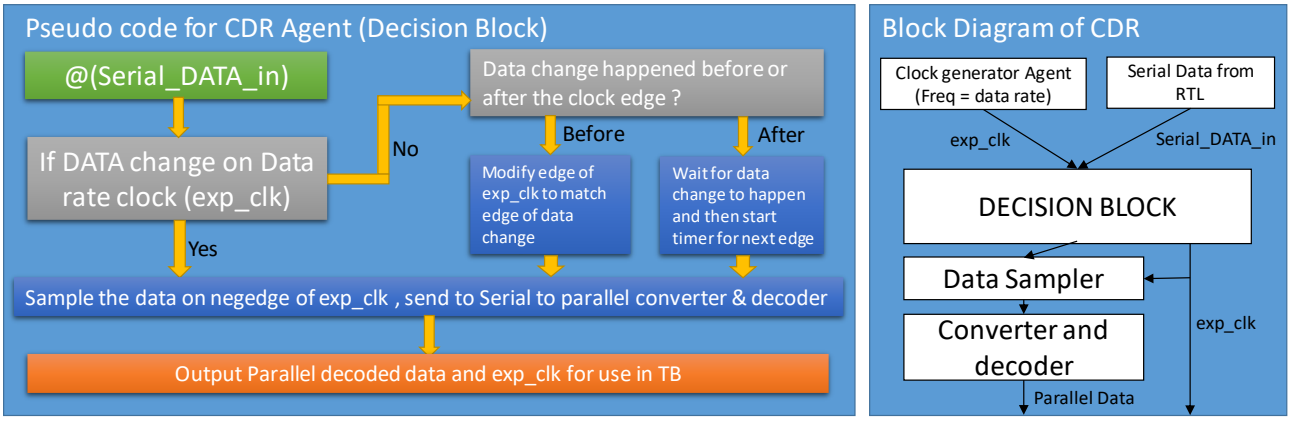

Figure 5 : Generic CDR flow and pseudo code

Figure 5 illustrates the implementation approach of the CDR agent. While employing a generic CDR algorithm, the concept remains adaptable to various CDR algorithms. Through sampling data based on an expected clock generated by the clock gen agent, deviations within expected JITTER and SSC values prompt realignment of the clock, ensuring data integrity.

III. Implementation and Application

The proposed UVC (Universal Verification Component) exemplifies seamless integration, offering an indispensable asset to diverse verification environments along with auto generated functional coverage bins to visualize during signoff. By harnessing UVM's packaging capabilities, integration becomes effortless. Simply importing the clock UVC package and instantiating the clock UVC environment facilitates access to all agent configurations, thereby enhancing RTL/TB functionality. The above is illustrated in *Figure 6*.

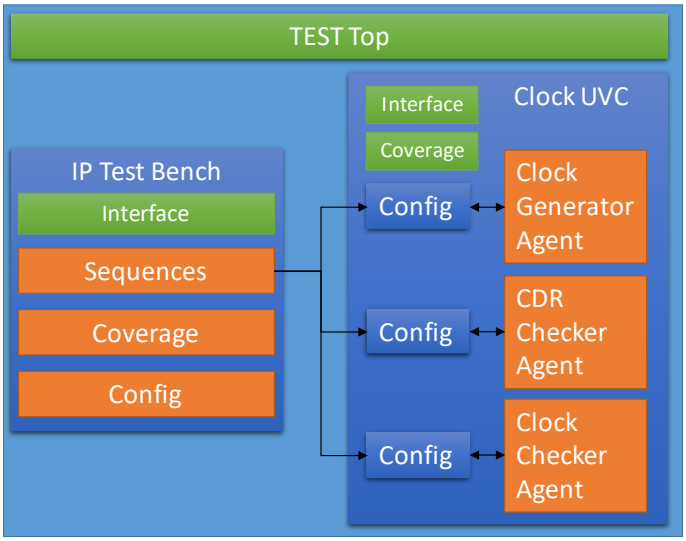

Figure 6 : Test Bench Architecture

Consider a practical scenario involving a PLL IP Verification. Initiating clock generation via the clock generator agent, tailored FREF clocks are effortlessly produced, their frequencies modifiable via TB sequence configuration.

Subsequently, the clock checker agent ensures the integrity of periodic signals, with configurable parameters enabling granular control over checks as shown in *Figure 7* & *Figure 8*.

```
clk_config = clock_param_cfg  :: type_id :: create("clk_config");
ev = uvm_event_pool::get_global("user_config_clk");
clk_config.round_off_digit = 1000;
clk_config.duty_cycle_value_arr[0] = 50;
clk_config.duty_cycle_value_arr[1] = 50;
clk_config.duty_cycle_tolerance_arr[0] = 1;
clk_config.duty_cycle_tolerance_arr[1] = 1;
clk_config.duty_cycle_tolerance_arr[2] = 1;
clk_config.clk_period_value_arr[0] = 1000000.0/`mct_intf.i_clk_freq;
clk_config.clk_period_value_arr[1] = 1000000.0/`mct_intf.i_apb_clk_freq;
clk_config.clk_period_value_arr[3] = 1000000.0/((`mct_intf.i_clk_freq/((
clk_config.period_chk_en[3:0] = 4'b0000;
clk_config.duty_cycle_chk_en[3:0] = 4'b0000;
clk_config.tolerance_en[3:0] = 4'b0000;
clk_config.clk_tolerance_value_arr[0] = 1;
clk_config.clk_tolerance_value_arr[1] = 1;
clk_config.clk_tolerance_value_arr[2] = 1;
clk_config.clk_tolerance_value_arr[3] = 1;
```

Figure 7 : Code snippet showing use of config based approach for programming the clock Checker UVC

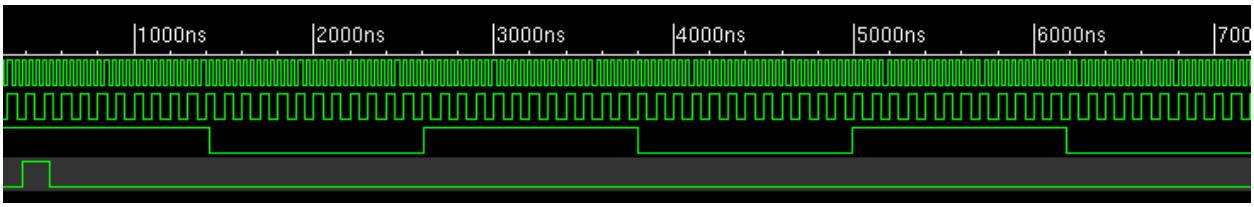

Figure 8 : Waveform snippet showing periodic signals of different frequency phase and duty cycle, all being checked by the clock UVC

The clock UVC can also be used to generate input clocks (FREF) to the IP. *Figure 9* shows the config based approach to program the clock generator agent. *Figure 10* shows a small pseudo code of clock generator agent with sinusoidal jitter. *Figure 11* shows the clock generator module generating output clock with sinusoidal jitter. The generator agent can be extended further to other kinds of jitter/SSC as well.

```
clk_config.jitter_sine_wave_freq[0] = freq;
clk_config.jitter_offset[0] = 0;
clk_config.jitter_ampl[0] = 1;
clk_config.clk_freq[0] = freq;
clk_config.jitter_en[0] = 1;
```

Figure 9 : config programming for clock generator agent

```
always @(dut_if.o_clk[i]) begin
  sine_wave_freq = config_obj.jitter_sine_wave_freq[i];
  time_s = $time/1000000000;
  sine_out = config_obj.jitter_offset[i] + (config_obj.jitter_ampl[i] * sin(2*pi*sine_wave_freq*time_s));
  if (config_obj.jitter_en == 0) sine_out = 0;
end
always dut_if.o_clk[i] = #((1000000.0/(2.0*config_obj.clk_freq[i]))+sine_out) ~dut_if.o_clk[i];
```

Figure 10 : Pseudo code for clock generator agent with sinusoidal jitter

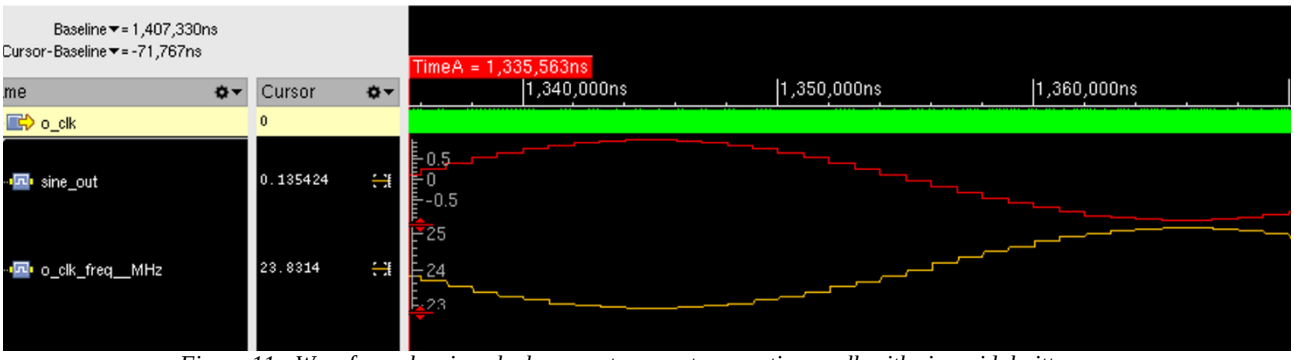

Figure 11 : Waveform showing clock generator agent generating o_clk with sinusoidal gitter

The primary objective of the CDR (Clock Data Recovery) agent is to extract serial data and clock signals, which are then transmitted to other agents or the testbench. The CDR agent operates in two key modes: clock data frequency alignment and clock data phase alignment. By configuring the cdr_mode setting in the configuration file, as illustrated in *Figure 12*, users can select either or both of these operational modes.
Figure 13 provides a pseudocode representation of the phase CDR operation, while *Figure 14* depicts the pseudocode for frequency CDR. Additionally, *Figure 15* presents a waveform demonstrating the application of phase CDR. This waveform illustrates how the actual TXDP lane of the serial IP and the exp_clk align with the data transition edges. Data is sampled at the negative edge of exp_clk to ensure stable data capture.

The three-agent approach described can be utilized both independently for their specific functions and collaboratively for a comprehensive verification process. For instance, if a specification requires that serial data maintain a PPM (parts per million) accuracy of ±200, the CDR agent can extract the expected clock rate at which the data transitions. This clock rate is then input into the clock checker agent, which will automatically determine if the specification is met. This streamlined process allows design and verification (DV) engineers to focus on connecting the agents rather than developing individual checkers for each feature.

Similarly, consider a requirement for a Phase-Locked Loop (PLL) IP, where the VCO must lock with an output frequency (FOUT) maintaining a ±200 PPM accuracy. In this scenario, the clock generator agent can be employed to create a sinusoidal jitter with an input frequency (FIN) exhibiting ±250 PPM variation. The FOUT should then reflect this variation accordingly. By using the clock checker agent, we can verify that the PLL achieves lock even with variations that exceed ±200 PPM, as sinusoidal jitter will occasionally produce values both below and above this threshold. This approach ensures that both the compliance with the specified requirement and the handling of negative scenarios are thoroughly examined.

```
clk_cdr_config.exp_data_rate = 10000000;
clk_cdr_config.cdr_en = 1;
// cdr_mode select mode of CDR ; [0] -> Frequency CDR ; [1] -> Phase CDR
clk_cdr_config.cdr_mode = 2'b11;
```

Figure 12 : config based approach to pass data rate to the CDR agent

```
fork
forever begin
        @(posedge cdr_en)
        @(tb_dut_if.r_ln0_txdp)
        exp_clk = 1;
        exp_clk_timestamp = $realtime;
        while(cdr_en) begin
                exp_clk = 1;
                #high_time;
                exp_clk = 0;
                #low_time;
                if (($realtime - exp_clk_timestamp) > (exp_period/2)) exp_clk_timestamp = $realtime;
        end
end
forever begin
        @(posedge cdr_en)
        while(cdr_en) begin
                @(tb_dut_if.r_ln0_txdp)
                data_change_timestamp = $realtime;
        end
end
forever begin
        @(data_change_timestamp)
        if (data_change_timestamp - exp_clk_timestamp > (exp_period/2)) begin
                exp_clk = 1;
                high_time = data_change_timestamp-exp_clk_timestamp-(exp_period/2);
                exp_clk_timestamp = $realtime;
                @(negedge exp_clk) high_time = exp_period/2;
        end
        else if (data_change_timestamp - exp_clk_timestamp < (exp_period/2)) begin
                low_time = (exp_period/2) + data_change_timestamp - exp_clk_timestamp;
                @(posedge exp_clk) low_time = exp_period/2;
        end
end
forever begin
        @(negedge exp_clk) extracted_data = tb_dut_if.r_ln0_txdp;
end
join_none
```

Figure 13 : Phase CDR Pseudo Code

```
forever begin
        @(tb_dut_if.r_ln0_txdp)
        curr_timestamp = $realtime;
        if (curr_timestamp != 0.0 && prev_timestamp != 0.0) begin
                new_frequency = 1/(curr_timestamp - prev_timestamp);
                if ((curr_timestamp - prev_timestamp) < 1.5*exp_period && (curr_timestamp - prev_timestamp) > 0.5*exp_period) begin
                        exp_frequency = new_frequency;
                end
        end
        prev_timestamp = curr_timestamp;
end
```

Figure 14 : Frequency CDR Pseudo Code

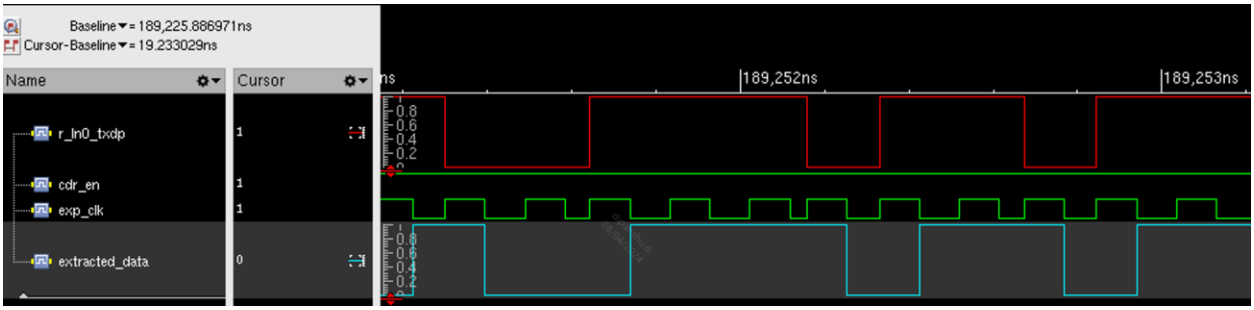

Figure 15 : Waveform showing exp_clk aligning to the tx edge and generating expected serial data (Phase CDR)

IV. Result And Conclusion

Extending beyond PLL IPs, the inclusion of the CDR agent renders the approach versatile, applicable across MPHY, USB, and various other PHY IP implementations. The clock generator assumes a pivotal role in generating diverse clock inputs, while the CDR block adeptly aligns with incoming data streams, particularly beneficial for encoded and serial data handling.

The versatility of the clock checker agent proves invaluable for inspecting periodic signal parameters, detecting anomalies such as glitches or X/Z states. This adaptability extends across projects and IP types, significantly augmenting value and utility.

Table 1 : Verification scenarios table

Verification Method	IP's Used in	Feature of UVC used
Clock Generation	All IP's	Input clocks, periodic signals and any other pattern needed for operation of IP
Clock Generation	MPHY, USB, PCIe & other PHY IP	Stress testing JITTER and SSC operation for Rx
CDR Agent	MPHY, USB, PCIe & other PHY IP	Extracting the data from serial lines and decoding data
Clock Checker	PLL IP's , MPHY & all other IP's	Verification of period , frequency, duty cycle on all periodic signal and glitch and X_Z checking on all output clock signals

Comprehensive verification, as depicted in *Table 1*, successfully identifies issues within PLL blocks, pinpointing bugs within RTL designs and uncovering glitches related to reset operations in various RTLs. Such thoroughness, facilitated by continuous signal monitoring, ensures robust verification across multiple project facets.

In conclusion, this paper heralds a groundbreaking paradigm, seamlessly integrating Clock Checker and CDR Agent alongside a versatile Clock Generator into the design verification process. All these benefits are attained with a minimal performance impact due to continuous checking of clock, typically around 5-15% in simulations. This can be further optimized by disabling and re-enabling the Clock Checker as and when required. Such an approach promises to meet the evolving demands of the electronics and semiconductor industry, where reliable and efficient designs reign supreme.

References

[1] "Make your Testbenches Run Like Clockwork!" by Markus Brosch (https://dvcon-proceedings.org/wp-content/uploads/make-your-testbenches-run-like-clockwork.pdf)

[2] https://web.stanford.edu/class/archive/ee/ee371/ee371.1066/lectures/Old/Older/lect_17_CDR_2up.pdf