2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Efficient Clock Monitoring System for SoC Clock Verification

Nam Pham Van, NXP, Munich, Germany (nam.phamvan@nxp.com)
Bernhard Braun, NXP, Munich, Germany (bernhard.braun@nxp.com)
Dirk Moeller, NXP, Munich, Germany (dirk.moeller@nxp.com)
Clemens Roettgermann, NXP, Munich, Germany (clemens.roettgermann@nxp.com)

***Abstract* — Clock systems belong to the most critical areas of the SoC design. It is therefore crucial to have accurate checks in place which verify the clock architecture with its requirements and constraints already during the design phase of a product. The presented approach demonstrates a Clock Monitor and Checker implementation which was developed to help with clock architecture verification in RTL and Gate-Level simulations (GLS). This paper also shows how these Monitors and Checker were used to verify functional clock requirements and clock constraints for the different modes of the SoC.**

Keywords—RTL design; RTL verification; Timing constraints; Clock monitoring; Clock verification;

I. INTRODUCTION AND MOTIVATION

Numerous causes of problems with clocks exist in today's SoC designs. Clock frequencies and duty cycles are required for correct protocol operation. The synthesis flow is based on clock constraints [1]. Safety architectures run self-test operations like MBIST and LBIST during normal operations, clocks are turned off or tuned for power reasons, etc. Due to those multiple reasons SoC test benches have to implement efficient clock monitoring and checking. The implementation should support different requirements in order to be regarded as "efficient".

All checks shall be on per default:
Instead of leaving it to the test case to enable a specific clock check they should be active per default in order to determine unexpected causes of violations.

Checks shall be dynamically controllable by SoC modes:
The monitors shall support to dynamically switch them on and off based on different SoC modes such as: functional mode, LBIST during functional operation, test-modes, availability modes etc.

Checks shall be controllable by test cases:
Test cases shall have the possibility to control the different clock checks as they sometimes violate legal conditions e.g. by forces.

It shall be easy and quick to debug and analyze fails:
This requirement was the main driver for the decision to replace legacy clock monitors and checkers with the new designed system as presented in this paper. We decided to implement a system in which the violating time stamps and the related clocks could be directly viewed in the waveform without the requirement to set breakpoints, analyze drivers of SystemVerilog (SV) events or handle debugging of dynamic classes. The proposed system is therefore based on static SystemVerilog module scope and still provides the controllability and flexibility as required. The "assertion fails" can directly be viewed in waveforms and tracing back of clock sources is straightforward.

Reusability and simplified integration:
The monitors shall be easy to integrate into the testbench. Monitors and checkers shall be independent from design and simulation tool. All checks shall have the same structure.

II. APPROACH

A. *Overview*

Figure 1 shows the structure of the clock verification for a System on Chip (SoC) design. The idea is to separate the complexity of the clock verification into layers and submodules. As can be seen in Figure 1, the concept consist three layers: the Clock Source Layer, the Clock Monitor Layer and the Checker Layer. The Clock Source Layer represents the SoC design with all the different clock domains and clock sources. The second layer consists a series of clock monitors that observe critical clock sources of the design. Thus, each clock source has a separated clock monitor instance which captures the high and low phase duration of a clock and provide these information to the third layer. The Checker Layer contains a series of Checker instances for each clock source. Base on the information of the clock monitor and the specified clock constraints the Checker performs the check. Each Checker instance itself consists four significant checks: min. Period Check, max. Period Check, Duty Cycle Check and expected Period Check and can be enabled or disabled when needed. The separation into layers simplifies the process of adding and removing new clock monitors and the corresponding checker.

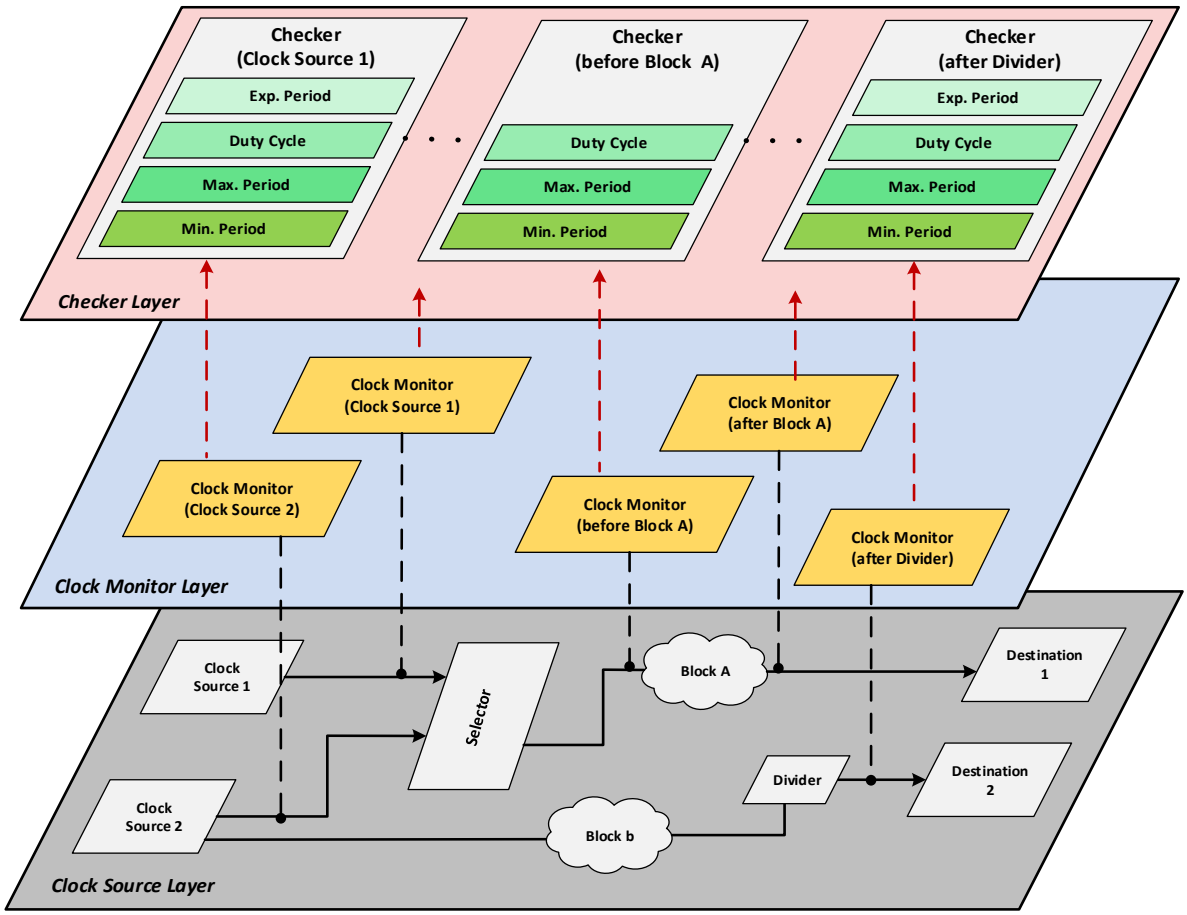


Figure 1 : Clock verification overview

B. Clock Monitor Module

The Clock Monitor Module is an independent module, which can be used to observe a specific clock source for verification purposes. It continuously provides the low and the high phase durations of a given clock. Each time a low or a high phase duration is calculated, a corresponding event signal is toggled to indicate the availability of the new data.

The Clock Monitor Module is split into two separate blocks, one each to handle either the positive or the negative clock edge (see Figure 2). Both blocks have identical structure and functionality. Furthermore, both blocks react on the negative edge of the reset signal.

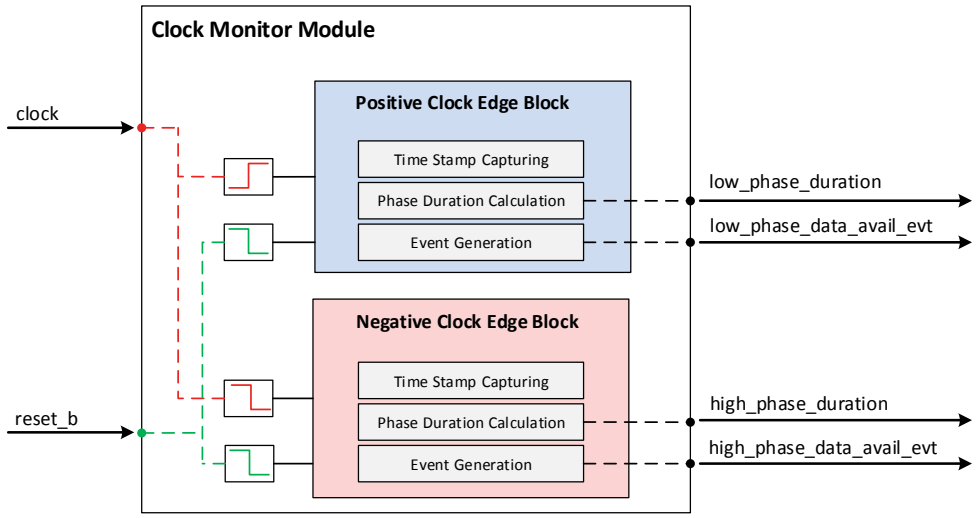
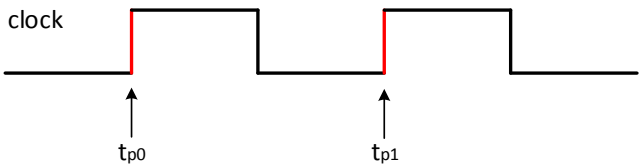


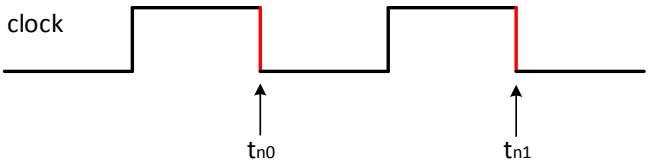
Figure 2 : Clock Monitor Module architecture

Each time a positive or a negative clock edge occurs, a time stamp is saved. For debug purposes and calculations, the current and the previous time stamp of each clock edge is retained (see Figure 3 and Figure 4).



Positive clock edge time stamp:
t_{p0} : previous time stamp
t_{p1} : current time stamp

Figure 3 : Time stamp capturing at positive clock edge



Negative clock edge time stamp:
t_{n0} : previous time stamp
t_{n1} : current time stamp

Figure 4 : Time stamp capturing at negative clock edge

The **Phase Duration Calculation** for a given clock is defined as follows:

- The low phase duration is the time between a negative clock edge and the directly following positive clock edge (see Figure 5).
- The high phase duration is the time between a positive and the directly following negative clock edge (see Figure 6).

For this implementation, the *low phase duration* t_{LP} is calculated with (1) and the *high phase duration* t_{HP} is calculated with (2).

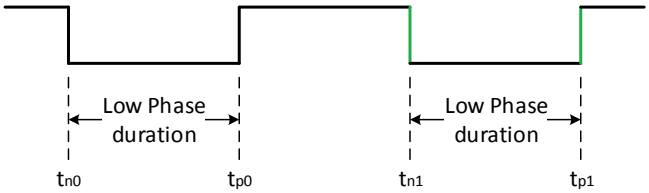


Figure 5 : Low phase duration

$$t_{LP} = t_{p1} - t_{n1} \qquad (1)$$

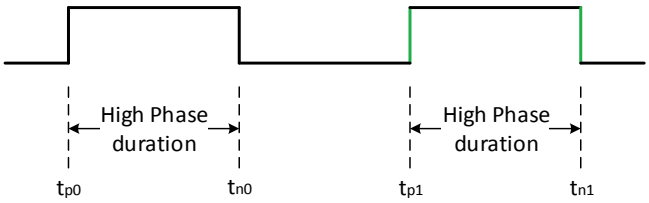


Figure 6 : High phase duration

$$t_{HP} = t_{n1} - t_{p1} \qquad (2)$$

The *low phase duration* t_{LP} is calculated only at a **rising** clock edge and the *high phase duration* t_{HP} is calculated only at a **falling** clock edge. The corresponding *low* or *high phase data available event* signal is toggled, *after* the low or the high phase duration value is calculated. Therefore, the *low phase data available event* toggles with every positive clock edge (blue waveform in Figure 7) and the *high phase data available event* toggles with every negative clock edge (orange waveform in Figure 7). In this example (Figure 7), the phase duration does not change because once the clock is stable, the low and high phase duration values are constant.

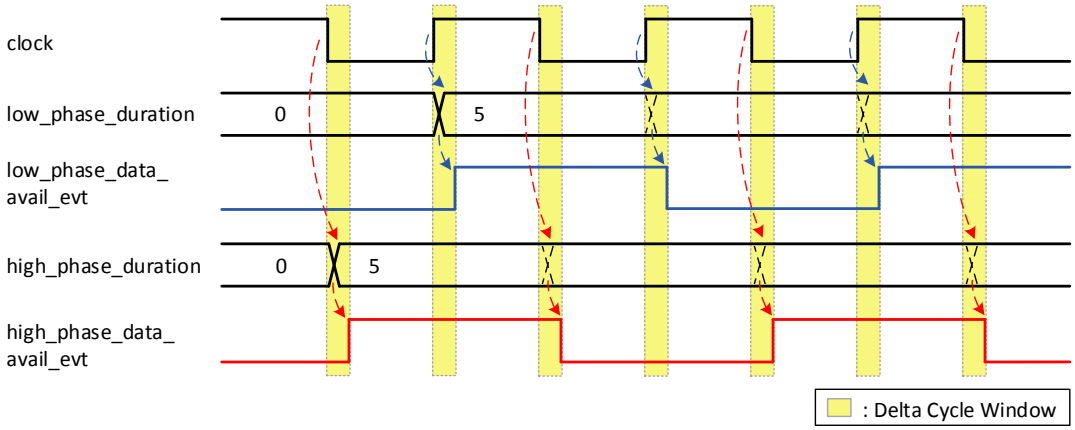


Figure 7 : Phase duration data event generation

C. Clock Check

Figure 8 shows the connection between the Clock Monitor Module and the Check Module. The Clock Monitor Module provides continuously the low and the high phase durations of a given clock. The input signals are *"standardized"* for all Check Modules. Based on the *low* or *high phase data available event* signal the Check Module identify new phase duration data. Beside the clock monitor signals, the Check Module has four additional inputs. The reset and enable inputs control the status of the Check Module and allows the activation and deactivation of the Check. The input *Input X* depends on the Check Module and defines the condition for the check (see Table 1). The input *Tolerance* is mandatory for every Check Module and defines the accuracy of the check.

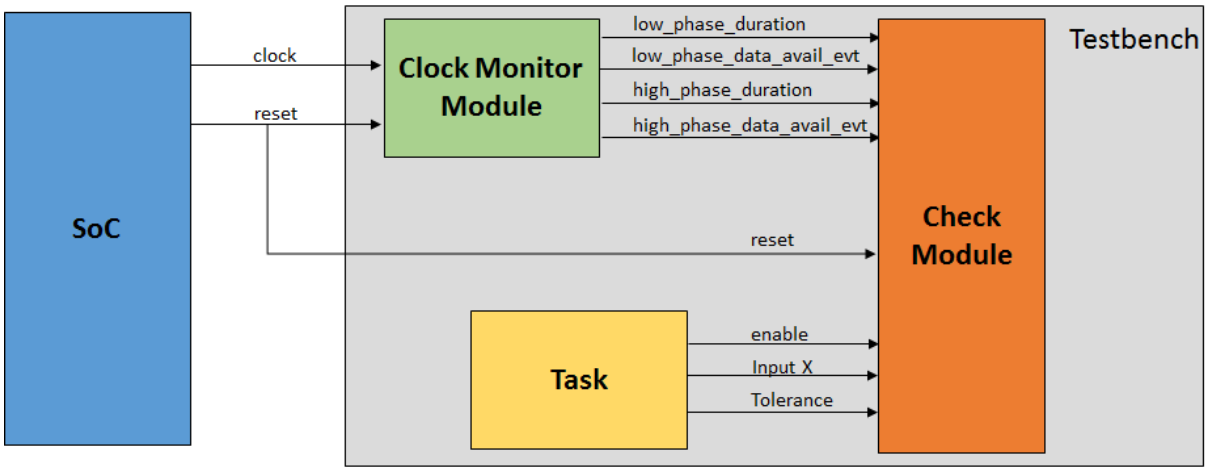


Figure 8 : Clock Monitor and Check Module structure

Check Module	Input X	Assertion Condition
Exp. Period	Exp. Period	Min. Exp. Period < Clock Period < Max. Exp. Period
Max. Period	Max. Period	Clock Period < (Max. Period + (Max. Period * Tolerance))
Min. Period	Min. Period	Clock Period > (Min. Period - (Min. Period * Tolerance))
Duty Cycle	Exp. Duty Cycle	Min. Exp. Duty Cycle < Duty Cycle < Max. Exp. Duty Cycle

Table 1 : Checks dependences

Clock Period = Low Phase Duration + High Phase Duration
*Min. Exp. Period = Exp. Period - (Exp. Period * Tolerance)*
*Max. Exp. Period = Exp. Period + (Exp. Period * Tolerance)*

Duty Cycle = Low Phase Duration / Clock Period
*Min. Exp. Duty Cycle = Exp. Duty Cycle - (Clock Period * Tolerance)*
*Max. Exp. Duty Cycle = Exp. Duty Cycle + (Clock Period * Tolerance)*

Furthermore, tasks were implemented to simplify the usage of the checks. Listing 1 shows the implemented checks as tasks and how they are called. When calling a task, parameters can be handed over to define the conditions of the check.

```
testbench.xosc_checker.max_period (.enable(1), .max_period(6.000ns), .tolerance(0.001));
testbench.xosc_checker.exp_period  (.enable(0), .exp_period(6.000ns),  .tolerance(0.001));
testbench.xosc_checker.min_period (.enable(1), .min_period(6.000ns),  .tolerance(0.001));
testbench.xosc_checker.duty_cycle  (.enable(0), .exp_duty_cycle(0.5),   .tolerance(0.001));
```

Listing 1 : Checks examples as Task

III. SoC Verification and Clock Monitor Generation

A. *Verification objective*

In complex SoC designs there are various clocks with different requirements and constraints. On the one hand there are functional clock requirements like minimum/maximum frequency or requirements on the duty cycle, which are mainly driven from application requirements (e.g. IP A must run with 80 Mhz and a 50% duty cycle clock). On the other hand there are clock constraints for the design and the physical implementation (synthesis) of the clock tree.

For these requirements it is crucial to check that there is no Over/Under-Testing and it needs to be ensured that Over/Under-Constraining is avoided. Otherwise the following could go wrong:

- ***Under-Testing:*** In this case, the test cases run too slow in an RTL or GLS simulation and therefore the maximum allowed frequencies are never achieved or tested.
- ***Over-Testing:*** The test cases run too fast in an RTL or GLS simulation and therefore the maximum allowed frequencies are exceeded or violated.
- ***Under-Constraining:*** The corresponding clock path is too slow and therefore the required frequencies cannot be met.
- ***Over-Constraining:*** It is more difficult to achieve the timing and additional buffers could be needed.

To ensure that none of these cases occur, it is important to verify the clock requirements and constraints thoroughly during SoC Verification. The clock monitors and checks which were introduced in the previous chapters are used for this Verification.

B. *Clock Monitor Generation*

Since there are numerous clock sources to monitor in an SoC design and to reduce the manual effort for the clock monitor implementation in the SoC Verification testbench, the clock monitors are generated using scripts (instead of creating them manually) based on the information from a ‘central clock database’.
The ‘central clock database’ contains all relevant clock information for the individual clock sources, such as hierarchical path of the clocks, permitted frequencies, duty cycle, etc. and are extracted from documents (e. g. Reference Manual, Datasheets, etc.). For each of the clock sources in the ‘central clock database’ a monitor and the corresponding checks are generated.

C. *Testbench integration and Pass/Fail criteria*

The checkers are implemented globally in the SoC Verification Testbench so that they run for each test case during regression runs. The PASS criteria is that the maximum frequency is not exceeded for all test cases, the minimum frequency is not underrun and the maximum frequency must be reached for at least 1 test case. Therefore, the following is checked during the regression:

- Did the Assertions really trigger?
- Did we achieve the required frequencies and were these frequencies not violated?

Only if the answer is ‘Yes’ for both cases, then the Regression is clean. If one of the answers is ‘No’ then the following Decision Matrix (Figure 9) can be used to find out if the test cases are wrong (if under/over-testing is done) or if the Requirements/Constraints are wrong (if under/over-constraining was done).

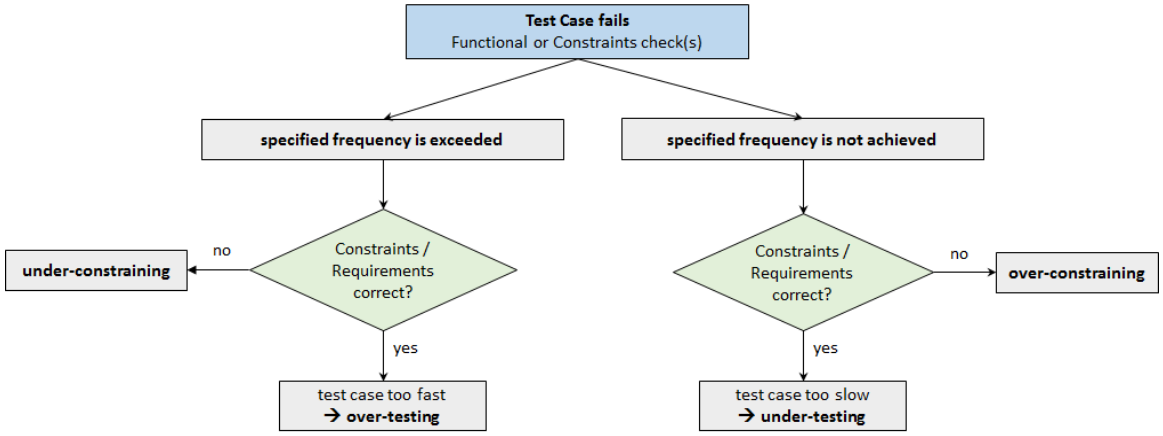


Figure 9 : Decision Matrix

All test cases, the functional checks as well as the constraints checks are getting and using the same frequencies. Additionally, different people deliver individual test cases and checks to cover special cases. Therefore, only if all checks PASS then all the items in Figure 10 are consistent.

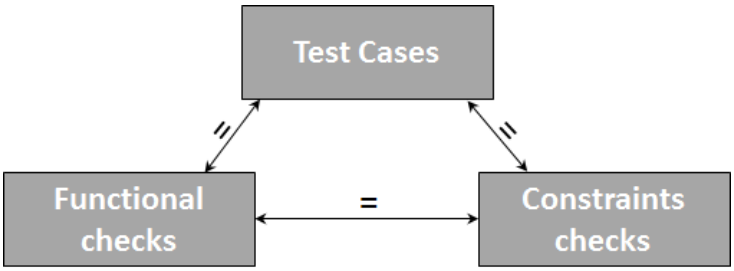


Figure 10 : Pass conditions

IV. Result and Conclusion

The paper presents a SystemVerilog clock monitoring approach. SV modules monitor and check the behavior of several clocks (dividers). A single source 'clock database' is used as input to automatically generate and instantiate the needed modules. The described clock monitoring strategy was successfully integrated into three projects and has proven the reusability. Over 200 clock monitors and checks were implemented for a current project and are per default enabled for all the test cases. The checks are controlled dynamically by SoC modes to ensure the right clocking in the corresponding mode.

Besides, test cases are able to enable and disable the checks manually in the stimuli to verify corner cases. The compact and clear interface, of the clock monitors and checks, makes it easy to add new instances for the testbench. The automated clock monitors and checks generation simplify the maintenance and keep it updated with design changes.

In all of the implementations so far, no noticeable increase of simulation time was noticed. Because of the very low resource demands it makes it ideal to include it into the regression run. Thanks to the implementation as modules, the debugging can be done with all available simulation tools and fails can be analyzed faster.

References

[1] G. Limmer, D. Moeller, M. Mueller, C. Roettgermann, “Validation of Timing Constraints on RTL, Reducing Risk and Effort on Gate-Level” NXP, submitted to DVCon Europe 2016.