2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Validation of Timing Constraints on RTL

Reducing Risk and Effort on Gate-Level

Peter Limmer, Dirk Moeller, Marcus Mueller, Clemens Roettgermann

NXP Semiconductors, Munich, Germany

{*peter.limmer|dirk.moeller|marcus.mueller|clemens.roettgermann*}*@nxp.com*

***Abstract*— In this contribution a specification-centered approach is introduced that is targeted on consistently capturing and transferring SoC design properties to gate-level design, while at the same time ensuring the validity of these properties through dedicated RTL verification. Essential aspects are covered, that influence the timing constraining - clock definition, asynchronous clock domain crossing and multi-cycle exceptions - along with respective methods to verify their validity on RTL. The presented approach demonstrates the opportunities of efficiently verify timing-related design properties on an abstraction level as high as possible.**

Keywords—RTL design; RTL verification; Gate-level design; Static timing analysis; Timing constraints;

I. INTRODUCTION

Static Timing Analysis (STA) is a vital method for ensuring, that a physical layout of a System-on-Chip (SoC) meets the timing requirements for all signal paths. Due to the existence of different clock-domains and asynchronous interfaces those timing requirements are often more complex than just "one clock period". Relaxing the timing of signal paths failing the STA allows the gate-level design to meet the timing requirements, but might lead to side effects. Obviously, manipulating a signal's propagation characteristics in relation to an associated clock signal requires the verification, that the logic functionality, as defined in the register-transfer-level (RTL) design, is not impeded. A common method to verify the timing behavior, among other things, is the gate-level simulation (GLS), which simulates the gate-level netlist with the appropriate timing information against the RTL test bench. A problem is the computational effort and the resulting long run-time of GLS [1,2]. Usually, this results in only a fraction of the tests being run on the gate-level netlist with a correspondingly low coverage (about 10%, according to the authors' experience). Furthermore, GLS is conducted very late in the design cycle and requires a high debug effort in case of detected failures, which puts pressure on the projects schedules. It is obviously beneficial to identify a significant portion of the problems as early in the design flow as possible. Problems rooted in RTL should be identified during RTL verification.

This contribution will focus on certain SoC design aspects that directly influence the definition of timing constraints. Since the required information is derived from RTL domain knowledge, it can also be verified on RTL to an as great extend as possible. This approach poses an effective and efficient supplement to early detect design issues, provide additional consistent information to support the physical design process, and thus prevent timing-related problems from emerging on gate-level. In the main part of this paper first a motivational example will be presented, illustrating a problem arising of discrepancies between RTL behavior and gate-level constraints in Section II. This motivates the establishment of a process defining the relationship between a central specification of essential RTL-to-Gate-level deliverables on one hand and both, STA constraint definition and RTL verification, on the other hand. This process is introduced in Section III, followed in Section IV by the presentation of the central aspects of clocking, asynchronous clock domain crossings and synchronous multi-cycle exceptions and their respective verification approaches on RTL. Concluding the paper in Section V, the benefits of the approach with regard to inter-process and team collaboration, as well as design and verification efficiency will be discussed.

II. MOTIVATION

The approach presented in this contribution has been developed gradually based on shortcomings identified during a number of real-life chip design projects. The following case is used as an example to illustrate possible problems and discuss the causes.

Consider the following SoC featuring a dual-core processor, a partial view of which is depicted in Figure 1. Each core is equipped with a data tightly-coupled memory (D-TCM), to which it has direct access. In addition, an external accesses, e.g. from DMA, are conducted via the bus interface unit (BIU), which is connected to the communication crossbar (XBAR). The core and D-TCM are part of the same high frequency clock domain, while the rest of the SoC, including the XBAR, lies in a low frequency clock domain. Both clocks are synchronous to each other with the high frequency clock being two times faster than the low frequency clock. The processor core was integrated as a hard macro.

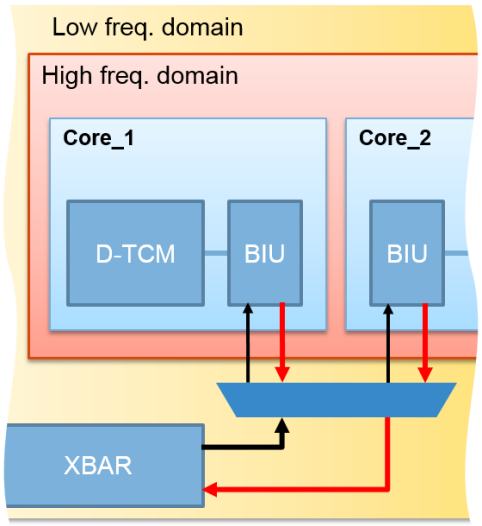

Figure 1: Partial view of SoC with data path crossing synchr. clock domain boundaries

The timing requirements in the high frequency clock domain were hard to meet. To avoid high speed paths from core to the rest of the SoC causing violations a multi-cycle exception of 2 was used to relax the read data path timing. The documentation of the core interface indicated that the BIU would perform a cycle adjustment to the low frequency clock. Though STA did not report any timing violation in this region, a GLS test case failed due to a timing violation in the XBAR, while performing a DMA read transfer from the D-TCM. The issue was traced back to the fact, that the BIU modified the read data unaligned to the low frequency clock edges, contrary to the documented cycle alignment feature.

The only choice to consistently solve the issue was to request (and wait for) an updated BIU logic with correctly implemented cycle adjustment handling, which required a re-synthesis of the processor core hard macro – an unfortunate development that late in the design process. On RTL verification side one reaction to this late finding was to specifically verify the correct cycle adjustment behavior of the BIU logic. This turn-around verification of this issue showed that the root cause of the timing problem could already have been detected on RTL, because the underlying assumption for the timing exception was verifiable on RTL.

Another observation was that the communication process between RTL design/integration and chip design teams was weak. The exceptions were informally exchanged via mail and reviewed before tape-out. An involvement of the verification team required to specify the assumption in a more formal approach, including specification tracing, and an extended RTL verification intent for functional SoC aspects with specific regard to clocking and cycle accuracy.

III. SPECIFICATION-CENTERED APPROACH

To mitigate the identified drawbacks a process was developed, that defines an interface between the responsibilities and the information created by front-end and back-end design teams, respectively. Agreeing on the artifacts to be exchanged between the teams is the first step to ensure consistency and integrity of the STA validation activities. Furthermore, the process facilitates RTL-based verification as an effective method to early detect issues that are risky and costly to deal with on gate-level.

A. *General process*

The central aspect of the process is the establishment of a common specification data base. Capturing *architectural and behavioral properties* of the SoC that influence the timing makes them part of the overall SoC specification document (STA_SoC_Guide) with a number of advantages. Like all other SoC specifications, also the timing specification is held under configuration management at a central location. It can be reviewed, discussed and developed by all involved parties, without knowledge getting lost. Furthermore, the items captured in the specification now represent *SoC requirements* that are individually tagged with specification tags. This makes them traceable throughout the design, integration and verification process.

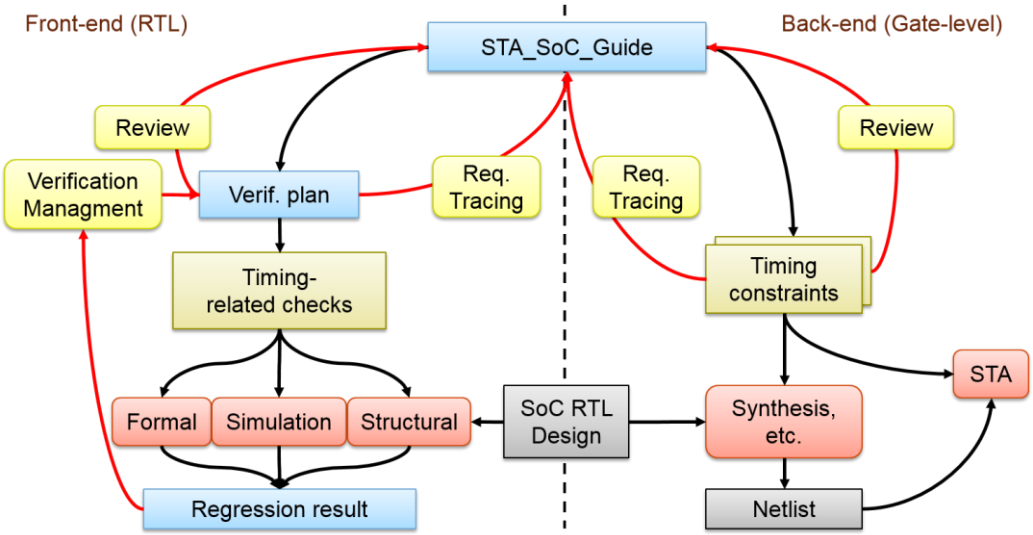

Figure 2: Specification-centered process for timing validation

As depicted in Figure 2, the STA-related specification influences two major chains of activities. On the back-end design side, according to the specification tags, a set of timing exceptions can be derived and implemented as *timing constraints* for both, gate-level synthesis & optimization, and static timing analysis (STA). On the front-end design side the specification items describe RTL requirements that are subsequently transformed into RTL *verification topics*. These are captured in the verification plan, from which the actual checks are implemented as part of the RTL verification environment and subjected to *structural analysis, simulation and formal analysis*. The verification success along the RTL regression is tracked according to the verification plan. For both, the creation of the timing constraints on gate-level, and the implementation of verification items, automated coverage checks are in place to ensure the *coverage* against the underlying requirements specification.

Deriving timing constraints on one hand and verification tasks on the other hand from a central abstract human-readable specification constitute two independent engineering steps to be undertaken by the respective experts with domain knowledge. The automated requirement coverage checks only ensure the existence of implementation related to a specification tag. The semantic relevance and correctness of the implementation against the specification still has to be validated by *expert reviews*.

B. *Capturing the Specification*

Since the specification is the central artifact that needs to bridge the process and team boundaries, a closer look on where this specification is coming from and what exactly needs to be captured will be taken here.

Special interest in timing behavior has always been peaking in cases of violated timing detected during optimization or STA, and in cases of failing GLS runs. Analyzing and debugging these issues led to the assertion of the required timing constraints the paths in question could be constrained with. These specific cases were reviewed against the RTL design to evaluate any impact on the functional correctness and, ideally, find justification for the correctness of the timing exception in the architectural and behavioral properties of the SoC. Eventually, the reasons for the established timing constraint were included in the STA_SoC_Guide data base as specification. This *bottom-up approach* has been driven by the need to handle necessary exceptions originating in findings on gate-level. The specification items served mostly as documentation between RTL and gate-level design teams for successful solutions.

Formalizing this, it can be stated, that the specification needs to contain two main aspects. The first is the *asserted behavior* of a signal path, from which the actual timing constraints can be derived. The second is an *SoC property*, which describes an architectural and/or behavioral characteristic of a set of interacting components. Relating to the example from Section II that means e.g., to formulate the specification of a multi-cycle exception:

*"IP A is running at **full sys_clk,** IP B at **half sys_clk**. The **ctrl** and **read data** signals must only be captured with the **rising edge** of the **half sys_clk**. Therefore, the **ctrl** and **read data** paths can be relaxed with a **multi-cycle of 2**."*

Essential is, that the SoC property serves as a *justification* for the asserted path timing – or, inversely, the SoC property allows for an *implication* towards the path timing assertion, as illustrated in the center of Figure 3.

This opens the path to generalization, due to the fact that an SoC property mostly affects multiple signals and paths, which often are subject to the same behavior. Thus, in a *top-down approach* utilizing RTL domain knowledge about architecture and functionality of the SoC, properties can be defined and implied path behavior can be derived from them. Implementing timing constraints based on specification obtained in this way helps avoiding timing violations, since paths get constrained based on a-priori knowledge before they might cause any violations. Top-down defined specifications also describe a valid space for possible timing relaxation to aid gate-level optimization and STA.

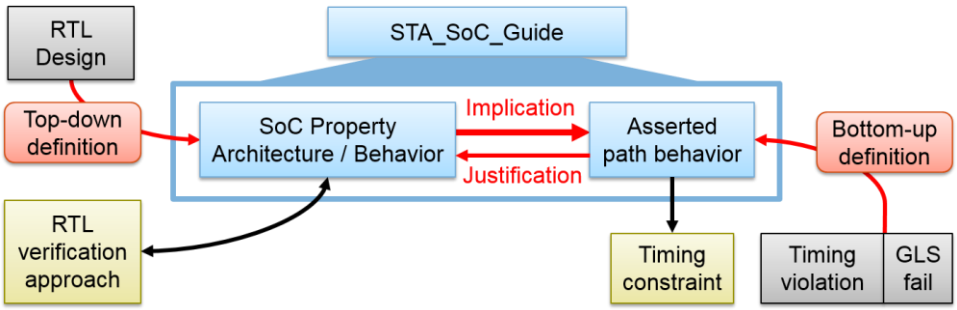

Figure 3: Capturing the specification

C. *Verification intent*

The approach introduced here mainly aims on *verifying the chain of causality* that allows the formulation of the intended timing exception. Verifying, that the specified SoC property holds and the implication is valid, allows to assume the validity of the signal path behavior. Based on RTL design intentions, the verification is possible on a higher abstraction level with a much larger scope that comprises properties with influence all across the SoC and throughout the operating modes. At the same time, the approach profits from utilizing existing methods of RTL verification, the full extent of an RTL test bench and the coverage achieved by RTL regression. Therefore, it constitutes an efficient and effective approach to supplement the gate-level verification of timing properties.

IV. RTL VERIFICATION APPROACHES FOR STA DELIVERABLES

Three major aspects are covered by the central specification – clock definition, clock-domain crossing and multi-cycle exceptions. In this section, the RTL domain knowledge leading to a consistent definition of deliverables to the gate-level design are discussed, along with the respective verification approaches. Instead of aiming on directly verifying any gate-level-specific aspects, they verify the SoC design properties and behavioral assumptions that cause the gate-level definitions that are derived from them to be assumed valid.

A. *Clock definition*

The clock domain definition consist of a complete set of properties of the primary and generated clocks of the SoC. The properties are divided into intra clock and inter clock properties, and an example illustration of these properties is shown in Figure 4.

The *intra-clock properties* for primary clocks are the minimum clock period (*min_pd*), the minimum low phase (*min_lo*) and minimum high phase (*min_hi*) durations, and the clock duty cycle (*duty*). For generated clocks two additional properties exists: the *source* clock (in most case a primary clock), and the generated *shape*. The *inter-clock properties* describe the relationship between two clocks. The first property is the frequency relationship, the second one is the *synchronicity* property, which specifies if the clock if synchronous or asynchronous to the other. For synchronous clocks, also the discrete phase relationship *p_lnc_cap* between launch and capture clocks has to be defined. The example in Figure 4 shows two primary clock sources PLL_A and PLL_B, which are asynchronous to each other, and the generated clock sources DIV_A1 and DIV_A2, which are synchronous, but with a phase relation to each other, as well as the generated clock source DIV_B1.

The clock properties are captured in a central clock data base, from which the clock constraint data for the synthesis and STA will be generated. It is essential that this specification is valid. To ensure this, RTL verification components are generated. The intra-clock properties min_pd, min_hi, and min_lo, as well as the inter-clock properties of synchronous clocks are monitored by a set of clock checkers. These clock checkers are attached to the clock generation components during RTL regression run and is verified that a specified properties a met for each test case.

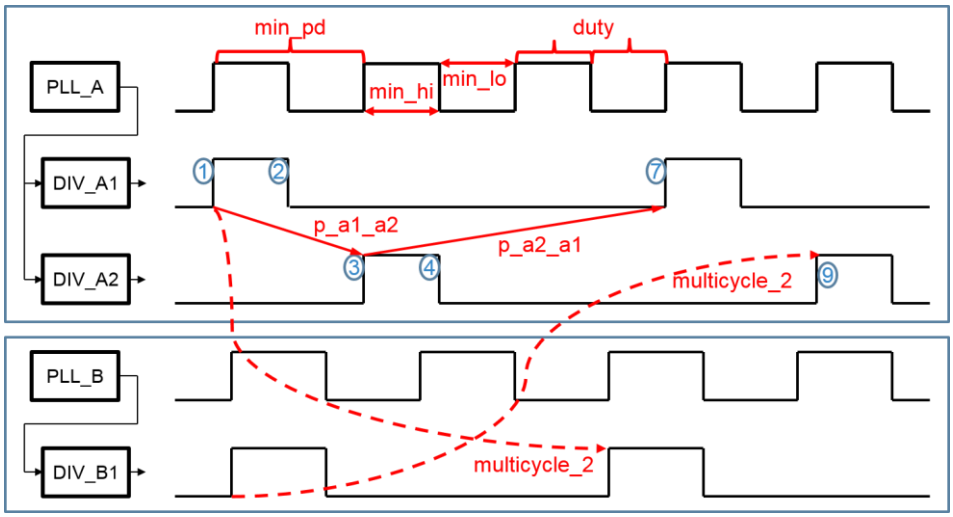

Figure 4: Intra- and inter-clock properties

B. *Asynchronous Clock Domain Crossing (CDC)*

If two generated asynchronous clocks, like the clocks A1 and B1 in the example in Figure 4 drive flip-flops that are communicating with each other, a dedicated clock domain crossing structure has to be implemented. Based on the clock synchronicity information in the clock specification data base, a setup for a structural CDC checker tool will be generated. This structural check and functional RTL verification regression ensure, that valid structures are implemented for all clock domain crossings.

In the synthesis process all paths between asynchronous clock domains will be considered as *false paths* for run time and performance reasons. Nevertheless, each CDC structure eventually requires *multi-cycle properties* to ensure correct functionality in silicon. These multi-cycle exceptions will be checked in a dedicated STA run, the setup for which is consistently generated out of the results of the CDC checker run.

C. *Multi-cycle exceptions*

Multi-cycle exceptions for optimization and STA can be specified for signal paths that do not have to meet the single cycle signal propagation time restriction relative to the clock(s) defined for the launching and receiving ends. Reasons to allow for multi-cycle exceptions can originate in special modes of operation of the SoC, component behavior and component interaction related properties. Two of these aspects are discussed in this section.

Implicit exceptions

Implicit multi-cycle exceptions relative to a given primary clock C appear, when two generated synchronous clocks C1 and C2 are specified with the divider values of c1 and c2 and the greatest common factor of both dividers is greater than 1, i.e. GCF (c1, c2) > 1. The clock definition provided to synthesis and STA defines these phase relationship via the edge definition, and synthesis as well STA will use a relaxed timing for the path. In Figure 4 the clocks A1 and A2 have a GCF (a1, a2) = 4. With the shown phase shift of one base clock cycle, an implicit multi-cycle of 3 for primary clock A a cycle appears between clocks A1 and A2.

The RTL verification has to proof that the specified phase relationship between the two clocks holds in the design in any case. The clock checker based RTL verification approach mentioned before ensures, that the waveforms meet the specification, that the verified RTL functionality is in fact based on the correct clock setup, from which consistent clock constraints can be provided to the synthesis and STA.

Operation mode dependent exceptions

Current SoCs feature a number of operation modes, which require the chip to behave different from the normal functional mode, e.g. device configuration, safety modes, low power modes, etc. The whole chip or parts of it may be required to run at different clock speed, various components could behave differently or even be shut down completely. Justification of multi-cycle exceptions may be based on identifying operation condition dependencies, or on the other hand the specification of operation mode and phase properties offers the potential to formulate a large number of multi-cycle exceptions.

Here, as an example for such a mode and the dependent exceptions the *device configuration phase* shall be discussed. While the device configuration is conducted the system clock is typically driven by a safe and reliable, and thus slow source, like an internal oscillator, instead of the full speed PLL source. The frequency ratio between these sources can easily be larger than 10 (e.g. 300 MHz full speed system clock frequency versus 16 MHz internal oscillator frequency). Structurally, the configuration sources and sinks (modules) are driven by the same system clock. However, according to functional knowledge the device configuration outputs can be assumed to be *stable* outside of the device configuration phase. Given these conditions, this allows to place multi-cycle exceptions with the ratio of full system clock speed and slow clock speed, which relax the timing of these paths significantly. The assumptions used as justification for those mode-dependent exceptions are properties, which can and have to be functionally verified on RTL.

Let M_{slow} be the mode of operation, where the system clock frequency f_{clk} is set to the low value f_{slow} (e.g. device configuration), in contrast to M_{fast} with the system clock running at f_{fast}. The RTL verification goals break down into two aspects – make sure the correct clock frequency is established for the respective mode, and the critical signals do only change in the intended mode:

1. $M_{slow} \rightarrow f_{clk} \leq f_{slow}$
2. $\forall s \in DeviceConfigSignals: \quad s = const \quad \leftarrow \quad M_{fast}$ or
   $\exists s \in DeviceConfigSignals: \quad s \neq const \quad \rightarrow \quad M_{slow}$

The first aspect is verified by *clock frequency monitors* which are tuned to the mode of operation. They are part of the simulation test bench and run in all test cases. The second aspects is covered by System Verilog *assertions* (SVA). These require formulating conditions on signals unambiguously identifying the intended mode, and mainly using the *$stable* construct in conjunction with the critical signals. Those SVA properties run as monitors in simulation and/or are proven or at least challenged via formal verification.

Protocol-driven exceptions in component interaction

Another opportunity for the application of multi-cycle exceptions are the connections between SoC components, as in bus communication structures, like e.g. AHB or AXI. Under the circumstances, that certain bus-connected modules run at lower clock speeds, and therefore can capture and assert data only in certain intervals, the timing of the paths to these modules can be relaxed. However, due to the "timelessness" of RTL simulation, it takes special measures to verify, that the data capturing takes place at the appropriate point in time and the intended logic functionality is still given. To correctly establish a multi-cycle exception the following key preconditions need to be met by design:

1. The launching and capturing clocks are synchronous;
2. The capturing clock frequency is an integer divide of the launching clock frequency;
3. Both, launching and capturing components are aware of the relationship between the clocks and the data exchange is handled correctly.

The RTL verification of the first assumption is covered by CDC checks, while the second assumption is verified by the clock monitor checks mentioned earlier. The verification of the third assumption has to be split into launching and capture check. The *launch/stability check* verifies that data launched from the fast clocked module towards a slow clocked one is asserted aligned to the slow clock edge and is stable throughout the transfer, i.e. one cycle of the slow clock. This could be verified with formal methods. At SoC level this formal approach might not be feasible, but the same assertions can be used as checks during RTL simulation regression. The following shows the SVA property describing this check:

```
// check that data only change with rising edge of slow clock
 property p_data_toggle;
  @(posedge fast_clk)
   ##1 $changed(data) |-> $rose(slow_clk);
 endproperty:p_data_toggle
```

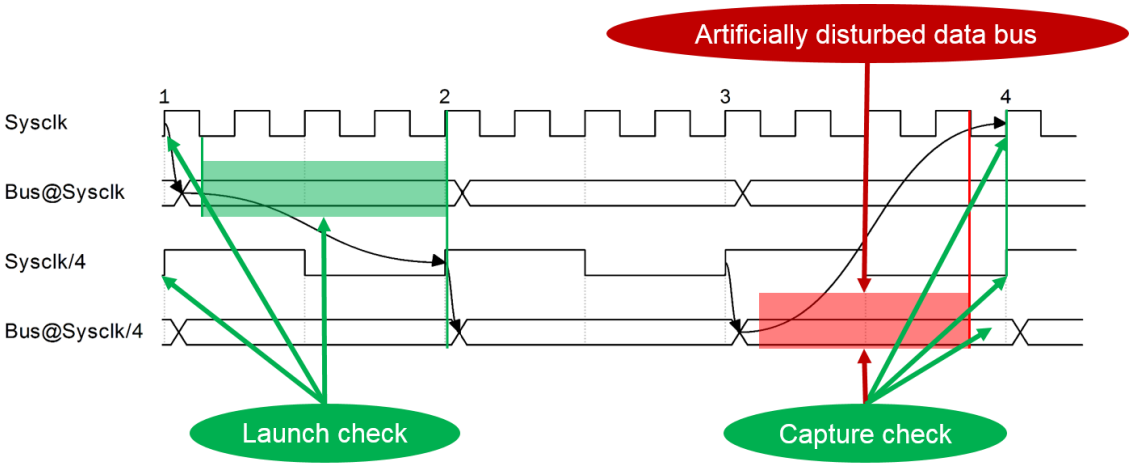

Figure 5: Verifying bus multi-cycle exceptions on RTL

The *capture check* verifies that data launched from a slow clocked module to a fast clocked one is only captured aligned to the slow clock edges. The method to verify this is to introduce bus disturbance in the simulation in the time frame the data shall not be captured by any bus component and check for data integrity. Figure 5 summarizes the two checks for a bus communication with a module clock frequency ratio of 4:1, for which multi-cycle exceptions of 4 have been formulated.

These are the RTL checks that subsequently verified the specific case described in Section II, and, in combination with the other presented approaches, have been in place to prevent similar issues for a number of design projects since.

V. CONCLUSION

In summary it can be stated that, based on shortcomings identified during a number of chip design projects, gradually a process has been developed and adopted, that established a consistent work flow between RTL design and gate-level design with the focus on timing constraints. The specification-centered design, review and verification process has effectively raised the awareness for timing issues and their resolution across the team and responsibility boundaries.

The added value of the presented approach is definitely the top-down definition of SoC properties for clock characteristics, CDC and timing constraints, and their respective verification. These presented methods effectively support the prevention of failures due to incorrect clock definition and incorrect timing exceptions, and also define a space for timing relaxation to aid optimization and STA. This proactively addresses root causes of timing problems, instead of reactively resolving issues arising late in the design cycle. Of course, claiming completeness is inappropriate – timing violations will occur, that have not been covered by any specification yet. Still, the process ensures their consistent capturing, the necessary constraining and the validation against the RTL design.

The developed approach is no replacement for gate-level simulation, but provides an additional measure to detect issues with timing influence earlier in the design cycle, even before the first synthesis run and before GLS has been implemented. Additionally, it profits from the larger coverage of the RTL regression compared to the time-consuming GLS, and from the lower debugging effort on RTL. Referring to the case study from Section II, the discussed fault was detected during GLS which ran only ca. 10% of the test cases of the RTL regression. It required about 4 days of debugging effort. The RTL fix required 1 day, along with more than 5 days for re-synthesis and physical implementation of the whole SoC, followed by the rerun of the gate-level regression for another day. Including the coordination effort between the involved teams it added up to more than 11 days of unscheduled project effort. In contrast to that the development of covering RTL assertions has to be seen, which took with 3 days and created negligible additional simulation effort within the RTL regression.

Due to relatively cheaply implementable preventive checks the risk of running costly recurring gate level debug, RTL re-design, re-synthesis and GL re-simulation cycles is reduced. The approach presented here demonstrates the opportunities of verifying design properties as soon as all the necessary information are available, with an appropriate and efficient method on an abstraction level as high as possible.

REFERENCES

[1] A. Khandelwal , A. Gaur, D. Mahajan, “Gate level simulations: verification flow and challenges”, EDN Network, May 05 2014, URL: http://www.edn.com/design/integrated-circuit-design/4429282/Gate-level-simulations--verification-flow-and-challenges, last accesssed July 15, 2016

[2] R. Goering, “Functional Verification Survey -- Why Gate-Level Simulation is Increasing”, Cadence Industry Insight Blogs, Jan 13 2013, URL: https://community.cadence.com/cadence_blogs_8/b/ii/archive/2013/01/16/functional-verification-survey-why-gate-level-simulation-is-increasing, last accessed July 15, 2016