A Cross-Level Verification Methodology for Digital IPs Augmented with Embedded Timing Monitors

# A Cross-Level Verification Methodology for Digital IPs Augmented with Embedded Timing Monitors

SARA VINCO, Politecnico di Torino, Italy
NICOLA BOMBIERI, University of Verona, Italy
DANIELE JAHIER PAGLIARI, Politecnico di Torino, Italy
FRANCO FUMMI, University of Verona, Italy
ENRICO MACII, Politecnico di Torino, Italy
MASSIMO PONCINO, Politecnico di Torino, Italy

Smart systems are characterized by the integration in a single device of multi-domain subsystems of different technological domains, namely analog, digital, discrete and power devices, MEMS and power sources. Such challenges, emerging from the heterogeneous nature of the whole system, combined with the traditional challenges of digital design, directly impact on performance and on propagation delay of digital components.

This paper proposes a design approach to enhance the RTL model of a given digital component for the integration in smart systems with the automatic insertion of delay sensors, which can detect and correct timing failures. The paper then proposes a methodology to verify such added features at system-level. The augmented model is abstracted to SystemC TLM, that is automatically injected with mutants (i.e., code mutations) to emulate delays and timing failures. The resulting TLM model is finally simulated to identify timing failures and to verify the correctness of the inserted delay monitors. Experimental results demonstrate the applicability of the proposed design and verification methodology, thanks to an efficient sensor-aware abstraction methodology, by applying the flow to three complex case studies.

CCS Concepts: • **Hardware** → *Simulation and emulation*; **Transaction-level verification**; *Timing analysis and sign-off*;

Additional Key Words and Phrases: Timing monitors; Code abstraction; SystemC TLM; Razor sensor; Verification

**ACM Reference Format:**
Sara Vinco, Nicola Bombieri, Daniele Jahier Pagliari, Franco Fummi, Enrico Macii, and Massimo Poncino. 2019. A Cross-Level Verification Methodology for Digital IPs Augmented with Embedded Timing Monitors. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, Article 1 (January 2019), 23 pages. https://doi.org/10.1145/3308565

## 1 INTRODUCTION

The design of smart systems has become challenging not only for its increasing complexity, but also for its emergent multidisciplinarity [8]. As a result, even though the design flow is highly

Authors' addresses: Sara Vinco, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy; Nicola Bombieri, University of Verona, Department of Computer Science, Verona, Italy; Daniele Jahier Pagliari, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy; Franco Fummi, University of Verona, Department of Computer Science, Verona, Italy; Enrico Macii, Politecnico di Torino, Interuniversity Department of Regional and Urban Studies and Planning, Turin, Italy; Massimo Poncino, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy.


1084-4309/2019/1-ART1 $15.00
https://doi.org/10.1145/3308565

standardized in the digital domain [43], the presence of multi-domain components in a single device arises novel challenges and design constraints, that affect interactions over the whole system. This requires novel and specific solutions, that must combine multi-disciplinary monitoring with good performance, still not affecting the overall design flow.

Most of the design constraints (e.g., frequency, power supply, temperature) are related to physical properties of the circuit, and thus require the insertion of suitable monitors. A typical example are delay monitors, used to detect and correct timing failures induced by manufacturing process variability, aging or temperature variations. All such monitors require to apply verification at low levels of abstraction, typically at RTL. Unfortunately, verification of physical properties related to specific design constraints at RTL has several limitations. First, simulation performance at RTL is prohibitive for reaching high-quality results, thus preventing an effective validation of the inserted sensors. Also, the manipulation of the RTL code for testing system correctness over different metric values is time consuming and not scalable to complex systems. Finally, verification of the RTL model, once integrated into a high-level system description of a smart system (e.g., a virtual platform implemented in SystemC TLM or C++) requires co-simulation instead of simulation, thus killing simulation performance of the whole system platform [22, 23, 45].

Such considerations on the performance of RTL and of its impact on both simulation and verification suggest that moving to system level might improve by far the verification process. Currently, the consolidated level of abstraction for system level design and verification is Transaction Level Modeling (TLM), that facilitates design space exploration and verification of the system without focusing on the implementation details. This guarantees a sound trade-off between simulation performance and accuracy for a wide range of user's needs during design and verification of digital components. The SystemC-based implementation of TLM has the additional advantage that it is deeply integrated within the SystemC framework, and that it can thus run simultaneously with models implemented with the Analog and Mixed-Signal (AMS) extension of SystemC. This allows to foresee a homogeneous simulation scenario for a multi-domain system.

In this context, tools for the automatic abstraction of existing RTL models represent a valuable support for the design of modern complex systems. Well-known examples are the RTL-to-SystemC abstraction tools for reusing RTL models of IPs [2, 21]. The automatic translation process to TLM preserves only the functionality of the original IP. On the other hand, most of the design constraints are related to physical properties of the circuit (e.g., frequency, power supply, temperature). As a consequence, the verification of the additional design constraints is not achievable, at the state of the art, at TLM.

Mixed-level modeling or co-simulation have been proposed over time to allow, for instance, design exploration and validation [37], RTL fault injections with error propagation at system level [33], and prediction of non-functional properties (such as aging) [28]. However, the lower level models act as bottlenecks in the mixed-level approaches, thus slowing down the simulation of the whole system.

This paper proposes a methodology for system-level verification of digital IPs augmented with sensors. Since many physical properties affect the timing of the digital IP, we adopted timing monitors, so that the effect of many physical properties can be captured concurrently.

This work builds upon [26], that introduced the proposed detection and correction paradigm. The new contributions are the following:

- an extended presentation of the design paradigm, focusing on the requirements on sensor features, on the sensors adopted in this work, i.e., the Razor sensor and the Counter-based sensor, and on the sensor insertion strategy;

- the extension of the RTL-to-TLM abstraction strategy, to allow faster simulation during the verification of the inserted delay sensors, to identify a tradeoff between accuracy and simulation speed, and to make the proposed approach more robust and applicable to more complex IPs;
- an extended presentation of the mutation analysis approach, that allows verification of the inserted sensors early in the design flow at system level;
- the application to three new case studies, chosen to prove the soundness and the applicability of the proposed approach to complex IPs.

The paper is organized as follows. Section 2 provides some background. Section 3 describes the implemented design and verification paradigm, that is deepened in Sections 4-7. Section 8 presents the experimental results, while conclusions are discussed in Section 9.

## 2 BACKGROUND

### 2.1 Detection and correction paradigm

The heterogeneity of smart systems, together with the coexistence of multi-domain subsystems in a single device, forces to extend the traditional digital design process to additional constraints on performance, reliability/robustness, power consumption and temperature [8, 50]. Such metrics can be monitored through specific physical quantities, i.e., frequency (propagation delay), supply noise, supply current, supply voltage and temperature. Their direct measurement, through the insertion of ad hoc sensors, allows to define a design paradigm based on detection and correction of the related constraints.

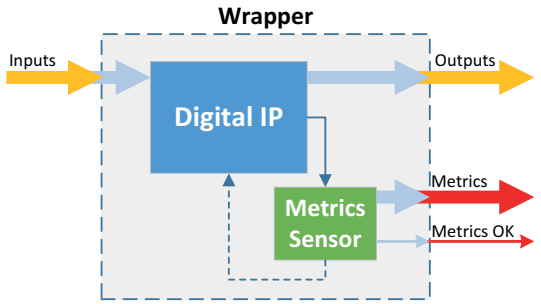



Fig. 1. Example of digital IP *augmented* with a sensor.

Consider a generic digital IP modeled at RTL through a hardware description language (HDL), as shown in Fig. 1. The IP is extended with a customized sensor, monitoring a specific metric and measuring the corresponding physical quantity. The sensor signals whether the related constraint is met (*Metric OK* output) and it may provide also a metric value (*Metric* output). This paradigm can be enriched with further functionality, depending on the target scenario:

- *detection only*: the sensor simply measures the quantity and signals whether it meets the specified constraint. Thus, *Metric* and *Metric OK* are the only outputs;
- *detection and correction*: the sensor provides an additional control signal that regulates some hardware knobs for “correcting" the corresponding metric value (the dashed line in Figure 1).

### 2.2 On-chip delay monitoring

With CMOS technology scaling, *delay variability* has become a critical issue. Indeed, a wide range of different physical phenomena result in alterations of the propagation delays of a CMOS circuit.

Manufacturing process variability caused by non-idealities (such as random dopant fluctuations and sub-wavelength lithography) makes the propagation delays non-deterministic and instance-specific [14]. Delay variations may occur among different realizations of the same circuit (global effects) as well as among different devices, e.g., standard cells, of the same circuit (local effects). Temperature variations across the die and so-called *hot spots* generate supply and threshold voltage fluctuations, which also result in alterations of nominal propagation delays [49]. In addition to the spacial dimension, these variations are also time- and context-dependant. Finally, aging effects such as Negative Biasing Temperature Instability (NBTI) or Hot Carrier Injection (HCI) cause nominal delays to drift during the lifetime of a device [3].

In order to cope with all these effects, increasingly large *design margins* (e.g., on the minimum supply voltage $V_{DD}$) should be included when constraining the traditional implementation of a CMOS device, to avoid timing failures even in worst-case variability conditions. Alternatively, occasional timing failures can be *detected* and *corrected*, rather than totally avoided, by following the paradigm defined in Section 2.1. This allows designers to reduce margins (e.g., use a lower $V_{DD}$) and therefore improve the metrics (e.g., energy efficiency) of the implemented circuit. Such constant monitoring of the timing behavior of a circuit is realized by means of on-chip *delay sensors* or *monitors* [15–18, 24, 38, 42, 48, 52].

On-chip delay monitoring architectures proposed in the literature mainly differ in the type of measurement performed: absolute measurement of delay values, or comparison w.r.t. a given threshold. In the former approach, the absolute measurement of path delay is achieved either using a Time-to-Digital Converter (TDC) to translate timing information into digital values [18, 38, 48, 52], or through time-to-voltage conversion to translate path delay into voltage levels [24]. In the latter approach, ad hoc sampling elements replace latches or flip-flops in the critical paths of the circuit. Occurrence of a timing violation is then detected by observing a signal transition within a given time window [15] or by performing a delayed comparison of the monitored signals [16, 17, 42]. Recovery mechanisms may also be implemented in order to correct the detected errors.

Once delay sensors have been embedded into a digital IP, their detection and correction characteristics must be verified. Several solutions based on fault injection can be found in literature [5]. Some techniques rely on simulator commands to easily manipulate model signals or variables without altering the HDL code. However, such commands are not standard, but rather simulator-specific. Other techniques modify the original RTL code, either by adding saboteurs in the design structure [41] or by modifying the behavior of some components by using mutants [4]. The main drawback is that both techniques require additional control signals to activate the occurrence of a fault and automatic tools to add/remove the HDL modification.

### 2.3 Mutation analysis

Mutation analysis relies on the concept of creating several models of the design under verification (i.e., a SW application or a HW IP model), each one *mutated* by introducing a syntactically correct functional change (i.e., a *mutant*) [19, 31, 39]. The purpose of such mutations consists of perturbing the behavior of the model, to verify whether the test suite is able to detect the difference between the original model and the mutated versions [36]. A transformation rule that generates a mutant from the original program is called a *mutation operator*.

Figure 2 shows the structure of a *mutation system*. When an IP model is submitted to a mutation system, the system first creates many mutated versions of the program (step 1), e.g., by replacing an operand with a different syntactically legal operand, or by modifying expressions replacing or inserting new operators, or deleting entire statements. Then, test cases are supplied to the system to serve as inputs to the program (step 2). Each test case is executed on the original program. If the output of a mutant program differs from the original (correct) output, the mutant is marked

as *killed* (step 3), otherwise it is marked as *survived* [27]. If after all test sets have been executed some mutants still survived, verification engineers can provide additional test inputs to kill such survived mutants, thus improving the test set quality.

If a mutant cannot be killed by any possible sequence of inputs, such a mutant is said to be *equivalent*. A model that has equivalent mutants is syntactically different but functionally equivalent to the model with no mutants. Automatically detecting equivalent mutants is impossible as such a model equivalence is undecidable [36].

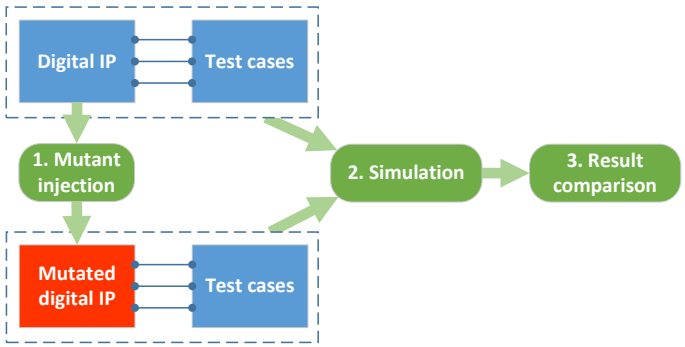



Fig. 2. Outline of the mutation analysis process.

Mutation analysis provides designers with an adequacy score, known as *mutation score*, which indicates the quality of the input test set. The mutation score is the ratio of the number of killed mutants over the total number of non-equivalent mutants. The goal of mutation analysis is to measure how far the mutation score is from 100%, which indicates that the test set is sufficient to detect all the design errors represented by the mutants.

Mutation analysis has been applied to languages for system-level design and verification such as SystemC [9, 32, 35, 44]. All these papers propose mutation models to verify the functional correctness of the SystemC and SystemC TLM descriptions, but none of such works aims at verifying timing constraints of the SystemC model through simulation.

### 2.4 SystemC TLM

Transaction level modelling (TLM) is a modelling style for implementing communication between IP blocks at different abstraction levels, each one with a different level of accuracy. It is applied to a variety of *use cases*, such as SW development, SW performance analysis, architectural analysis, and HW verification.

Rather than defining an abstraction level around each use case, the OSCI TLM-2.0 standard defines a set of *interfaces* (i.e., blocking, non-blocking, direct memory, and debug interfaces) and provides a library of *primitives* (e.g., `b_transport()` and `nb_transport()`) for implementing the communication side of transaction-level models. The standard specifies a number of *protocols* that are appropriate for, but not locked to, the various use cases.

The different TLM protocols allow designers to describe and simulate a design with different levels of detail. The best-suited protocol is selected depending on the target use case and each protocol is implemented by using specific TLM primitives.

The most adopted TLM-2.0 protocols are loosely-timed and approximately-timed. *Loosely-timed* (LT) is appropriate for software development, by using, for example, a virtual platform model of

an MPSoC, where the software may include one or more operating systems. Models implemented with this protocol have a loose dependency between timing and data. They do not depend on the advancement of time to be able to produce a response and, normally, resource contention and arbitration are not considered. *Approximately-timed* (AT) is appropriate for architectural exploration and performance analysis. Models implemented with this protocol have a much stronger dependency between timing and data, as AT forces models to synchronize the transactions before processing them, thus triggering multiple context switches in the simulation, eventually resulting in performance penalties. On the other hand, they easily model resource contention and arbitration. It is important to note that both the protocols are independent from the underlying SystemC management of time, that can be simulated only with artificial `wait()` invocations.

## 3 PROPOSED FLOW

Given the crucial role of propagation delay in design correctness, and its strong dependency from physical phenomena, the embedded monitors adopted in this work are *timing sensors*. Digital IPs are thus augmented with delay sensors, that are automatically inserted at RTL to monitor the IP timing characteristics: by sensing the propagation delay in appropriate locations of the digital IP, performance can be monitored and optimized to improve circuit reliability.

Once that sensors have been inserted to detect delays and temporal behaviors in the RTL IP, a verification step must be applied, to verify their effective detection of circuit delays. State of the art approaches work at RTL, either by applying fault injection or by adding saboteurs to the original IP. However, this makes the already slow RTL simulation even more time consuming.

This work proposes to move the simulation phase to TLM, with the goal of speeding up simulation. The RTL IP, enhanced with the delay sensors, is thus abstracted to TLM by using state of the art techniques. Given the timeless nature of TLM, delays must be expressed as modifications of the IP code: this is achieved with the definition of mutants, and mutation analysis is applied to evaluate the effectiveness of the detection and correction mechanism.

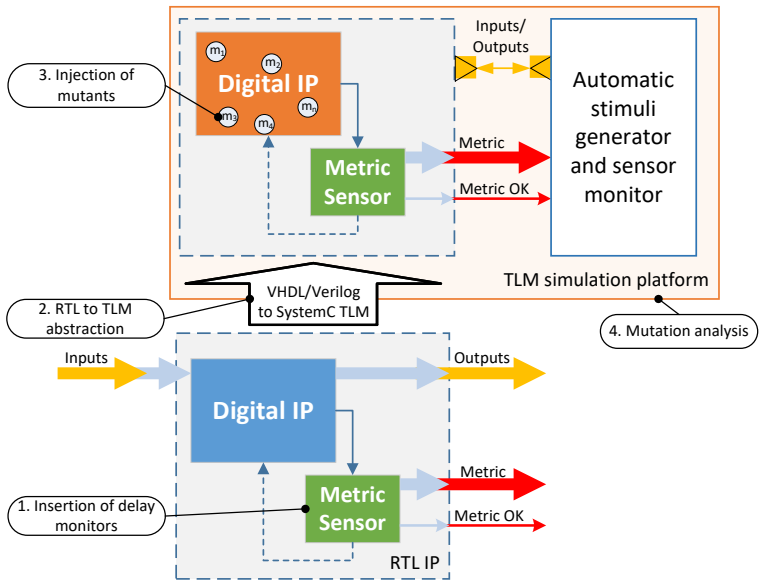



Fig. 3. Overview of the verification methodology.

The resulting methodology is made of the following steps, depicted in Figure 3:

(1) *Insertion of delay monitors.* The delay monitors must be inserted on the critical paths: it is thus necessary to identify delay monitors suitable for the proposed methodology, and to define a strategy to identify critical paths to be monitored (Section 4).
(2) *RTL-to-TLM abstraction of the augmented digital IP.* The RTL model of the digital IP, enhanced with sensors, is abstracted to SystemC TLM for fast simulation (Section 5).
(3) *Injection of mutants in the abstracted digital IP.* Delays do not exist at TLM, that abstracts timing: they are thus modeled as modifications of the IP code, called mutants, that are automatically injected in the abstracted digital IP (Section 6).
(4) *Mutation analysis.* The abstracted and injected digital IP and sensor are connected to a stimuli generator, which activates each mutant to test the detection and correction mechanism (Section 7).

## 4 INSERTION OF DELAY MONITORS

This Section presents the delay sensors adopted in this work (Section 4.1) and the strategy followed to automatically insert them in the starting IP (Section 4.2).

### 4.1 Delay sensors

The detection and correction sensors constitute an additional functionality w.r.t. the starting IP. Three essential requirements must be met for the sensor implementation to guarantee a correct adoption of the proposed paradigm:

- sensors must be synthesizable;
- sensors must be digital and should carry out no A/D conversion;
- the monitored quantity must affect either functionality or timing of the IP on the circuit path, and sensors must highlight any violation of the monitored property at the functional level, e.g., through output ports.

These conditions guarantee that the information is preserved also after the conversion to a high-level model at TLM. At higher abstraction levels, the IP model is less accurate and preserves only timing and functional information. Thus, if the monitored physical phenomenon (e.g., temperature, aging) is visible through its effects on functionality or timing of the IP, its effect is still visible and replicable at higher levels of abstraction. This is straightforward for delay monitors, as delay impacts both timing and functional behavior of the IP. Given that delays are a side-effect of many physical phenomena, this paper focuses solely on delay monitors. Note that the sensor may be embedded in the IP itself, i.e., it does not need to be external and distinguishable.

This paper proposes two different architectures for an on-chip delay sensor meeting such requirements: an extended Razor flip-flop (Section 4.1.1) and a Counter-based monitor (Section 4.1.2). The sensors can be used either individually or in combination, depending on the required level of precision, i.e., either a generic fail/no fail information or a more quantitative information on system delay.

*4.1.1 Modified Razor flip-flop.* The first monitor implementation is based on the Razor flip-flop (FF) concept [16], selected to obtain a timing failure detection and possibly correction within a few clock cycles.

The architecture of the Razor sensor is depicted in Figure 4.a. The IP is a generic IP, featuring one input signal (`DATA_IN`) and one output signal (`DATA_OUT`), and it contains three critical paths, labeled by `C0`, `C1` and `C2`. The key idea to monitor delay on such critical paths is to augment the functionality of a standard flip-flop (FF) with the capability of self-recovery in presence of a failure. This is achieved by replacing the standard flip flops at the end of `C0`, `C1` and `C2` with Razor sensors.

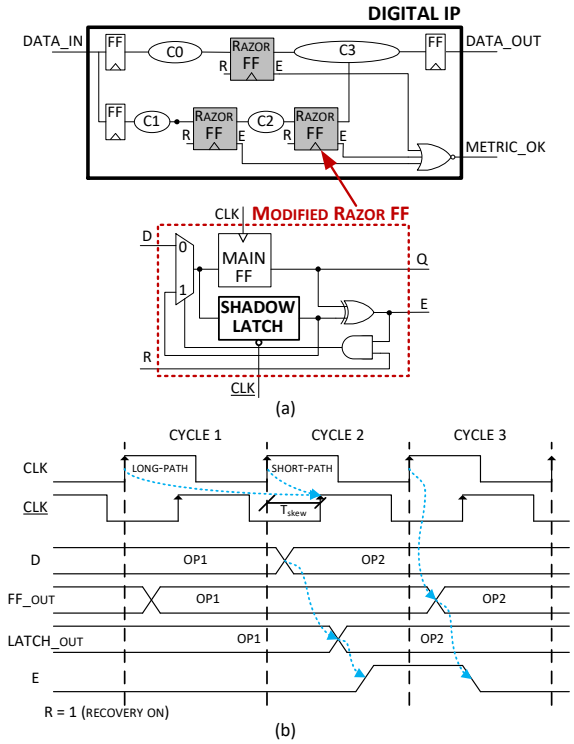



Fig. 4. Example of digital IP augmented with the modified Razor replacing FFs of critical paths (a) and timing diagram of the modified Razor sensor mechanism.

As a result, the digital IP is extended with a new output port (`METRIC_OK`), used to notify detected delays.

In detail, the Razor sensor enhances the FF by introducing a *shadow latch* that samples the FF input data on the negative level of the delayed clock signal $\underline{CLK}$. The $\underline{CLK}$ is delayed by half *CLK* period. If the values contained by the FF and by the shadow latch differ, an error signal *E* is asserted to notify the timing failure.

The correction feature can be selectively activated on each modified Razor FF acting on the corresponding signal *R*. The Razor flip-flop has been modified with an additional multiplexer for allowing the self recovery mechanism: when the control signal *R* is high, the recovery mechanism is executed and the error in the faulty FF is corrected.

*Working mechanism.* Figure 4.b details the behavior of the Razor sensor:

- *correct timing*: in a given clock cycle (cycle 1), if the combinational path C1 meets the setup time for the main FF, then both the main FF (positive edge triggered) and the shadow latch (later activated on the negative level of *CLK*) will latch the correct data. Therefore, the XOR gate's output is 0 and there is no error to flag at the output E (i.e., *E = 0*);
- *timing failure detection*: if the timing constraints on the combinational path C1 are not met because of variability issues, e.g., aging, temperature, process variations (cycle 2), the main FF will latch an incorrect data (i.e., *OP1* instead of *OP2*). Still, the shadow latch will latch the correct data (*OP2*), thanks to the delayed operating mode. As a consequence, the output of the XOR gate will rise to 1 and the error is detected and flagged in output (*E = 1*);

- *timing failure detection and correction*: if the error correction mechanism is enabled (cycle 3, *R = 1*, AND gate transparent), the multiplexer connects the latched value with the input of the main FF. In the subsequent cycle, when the recovery phase is executed, the normal operating mode of the system must be delayed of a cycle by the adoption of various strategies, e.g., by interrupting the normal pipeline operation.

*Sensor characteristics.* Since *CLK* is delayed by half *CLK* period, the Razor *working time window* is bounded by the rising and falling edge of CLK. The *area overhead* of a modified Razor FF is quite modest, as it is about one standard FF. In addition, the set of critical paths to be monitored represents a small percentage of the overall paths in the circuit.

*Adherence to paradigm constraints.* The Razor sensor satisfies the paradigm constraints. It is entirely digital, as it was designed to operate synchronously w.r.t. the augmented IP, and it is synthesizable. Therefore, it can be fully abstracted to SystemC TLM. Furthermore, the sensor affects the functionality of the augmented IP, both through the error notification process (output *E*) and the presence of both a FF and a shadow latch, that highlight the possible presence of a delay through its effect on the stored values.

*4.1.2 Counter-based monitor.* The second monitor implementation relies on a simple counter to measure the propagation delay on critical paths of the digital IP (Figure 5.a). The original IP is the same as for Figure 4. Compared to the modified Razor FF, the IP extended with Counter-based monitors provides also an absolute measure of delay rather than a simple timing failure detection (`MEAS_VAL`).

Reference and measurement values are computed by referring to a higher frequency clock (i.e., *HF_CLK* in Fig. 5.b), whose frequency is multiple of the main clock frequency (i.e., *MAIN_CLK*). Using the additional clock *HF_CLK*, the monitor enumerates the amount of *HF_CLK* periods elapsed for the signal propagation from the path start point to the path end point and thus can provide a measurement of delay.

The measurement is performed during a predefined time window called *observability window* (i.e., *OBS_WIN* in Fig. 5.b) where all signal transitions are captured. The position in time and width of *OBS_WIN* are chosen at design time according to the expected time interval where signal transitions may occur.

Two registers (*R1* and *R2*) store the counter value on the occurrence of both rising and falling transitions. The delay measure is then selected according to the last captured transition. A control block compares the obtained value with reference values determined at design time.

*Working mechanism.* Figure 5.b details the behavior of the Counter-based sensor:

- when *OBS_WIN* rises to logic 1, the measurement process terminates on the last transition of the current monitored path signal *CURR_CPS*, which can be either positive or negative. A positive transition activates the enable signal of the register *R1*, in order to sample the actual counter output value. Similarly, a negative transition activates the enable signal of the register *R2*, so that the actual counter output value is stored in this register. Therefore, register *R1* and *R2* operate in a mutually exclusive way. Depending on the transition type, the digital measure *MEAS_VAL* is continuously updated, without resorting to ad hoc transition detectors;
- when *OBS_WIN* falls to 0, the registers hold the last rising/falling transition, whereas the latches hold the last value assumed by the signal in the window, which is then used to select the correct register output value;

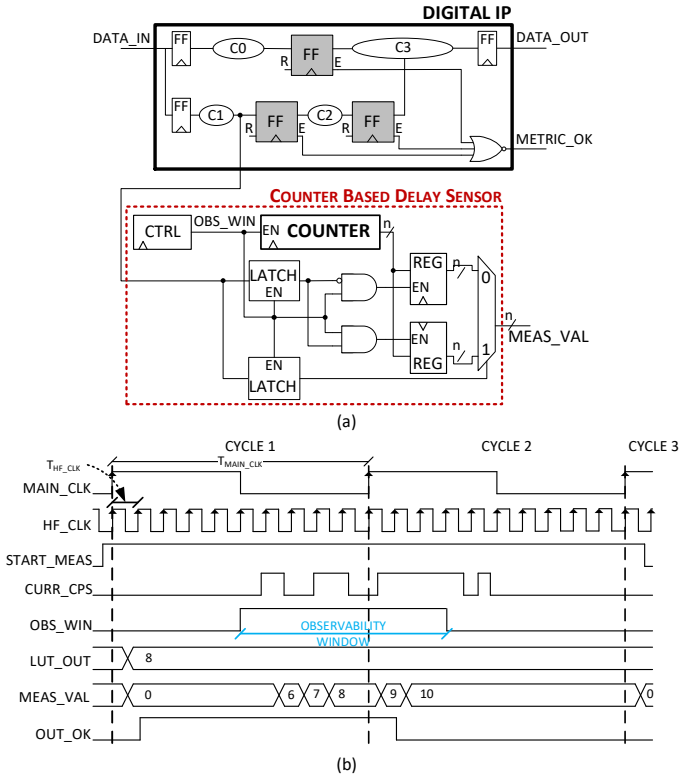



Fig. 5. Example of digital IP augmented with the Counter-based monitor connected to critical path end points (a) and timing diagram of the Counter-based sensor working mechanism (b).

- the reference path delay is then compared with the measured delay value and, if the timing constraint on the current path under observation is met, *OUT_OK* assumes logic value 1;
- the circuit output values are stable for the next *MAIN_CLK* rising edge (cycle 3) in order to be captured outside, while internal registers are reset to 0.

*Sensor characteristics.* In general, the main sensor characteristics depend on the *HF_CLK* period. For each critical path, the *latency of measurement* is three *MAIN_CLK* cycles. Once a measurement on a critical path is accomplished, another measurement process begins on the next *MAIN_CLK* rising edge, so that after three *MAIN_CLK* edges the delay of the next path can be read out. This procedure is repeated for all critical paths that have to be measured. Once all path delays have been measured, the circuit goes back to idle state.

Since the counter is synchronous with respect to *HF_CLK*, the *maximum resolution* is the *HF_CLK* period, and the *maximum error* is half of the *HF_CLK* period.

The *detectable delay range* depends also on the observability window: the maximum measurable delay corresponds to the time interval beginning with the first *MAIN_CLK* rising edge (signal transitions start to propagate through the monitored path) and ending with the falling edge of *OBS_WIN* (no more signal transitions are captured).

The *area overhead* is limited. As an example, in the case of 10 monitored paths and measurement bit width 8 bits, the occupied area is equivalent to approximately 352 NAND2 gates (minimum size of NAND in the library). Consider that a fast 8x8bit (16 bit dynamic range) FIR filter has a size of

about 10700 equivalent NAND gates, thus, the Counter-based monitor would increase the FIR size by 3.3%.

*Adherence to paradigm constraints.* The Counter-based sensor satisfies the paradigm constraints. It is entirely digital and it was designed to operate synchronously w.r.t. the augmented IP. The sensor has been successfully synthesized with Synopsys DC using a 45nm standard cell library, therefore it can be fully abstracted to SystemC TLM. Furthermore, the sensor extends the functionality of the augmented IP and it makes the presence of a delay visible through the *MEAS_VAL* output and the *R1* and *R2* registers.

### 4.2 Delay sensors insertion strategy

In order to contain the area and power overheads associated with the detection and correction paradigm described in Section 2.1, delay sensors should only be inserted at the endpoints of "critical" timing paths [16]. More formally, at the considered voltage-frequency target(s), those endpoints that would not incur setup-time violations even in worst case conditions (of local and global process and temperature variability, aging effects, etc.) do not require sensor augmentation.

Consequently, the identification of insertion locations for delay sensors requires a back-annotated analysis of the RTL model, using timing information obtained from a first (sensor-free) synthesis of the IP. The maximum propagation delay of each path is then determined using a *Static Timing Analyzer* (STA). Herein, the term *static* in STA is used to underline that the analysis performed in this phase is not based on simulation (*dynamic*), not to refer to any particular algorithm. Indeed, our methodology is agnostic of the specific timing analysis algorithm utilized, as long as the latter can identify suitable sensor locations in a conservative way (i.e., paths that are left without sensors must have a timing violation probability very close to zero). The identification process can therefore leverage any of the advanced variations of deterministic and statistical STA proposed in the literature, such as those accounting for path correlations, process variation, etc. [30, 46].

The easiest way to separate critical and not-critical paths (in case of a standard deterministic STA) is to adopt a *threshold-based* approach: all those paths whose critical setup *slack* is smaller than a threshold (in ps) are binned as critical, and vice versa. The threshold can be determined using standard design margin considerations, based on the relevant physical sources of variability. Industrial-level STA tools are already equipped with all the necessary features to compute such threshold, i.e. multiple process-temperature corners analysis, aging and local *On-Chip Variation* (OCV) modeling, etc.

Once critical paths have been identified, one sensor is inserted at each of these paths endpoints, by means of automatic modifications of the RTL model. Specifically, the RTL *signal* corresponding to the target endpoint is connected to a newly created instance of the delay sensor component (Razor or Counter-based), possibly through an intermediate variable used to extract single critical bits from a multi-bit signal. New *ports* are also added to the top-level IP model, for the connection of the support clocks (*CLK* and *HF_CLK*) and of the delay sensor outputs.

Although not strictly related to the proposed flow, notice that sensor locations should be remembered for the following post-verification synthesis of the IP, since they require special care during implementation (e.g., to avoid short-path issues in the case of Razors [16]).

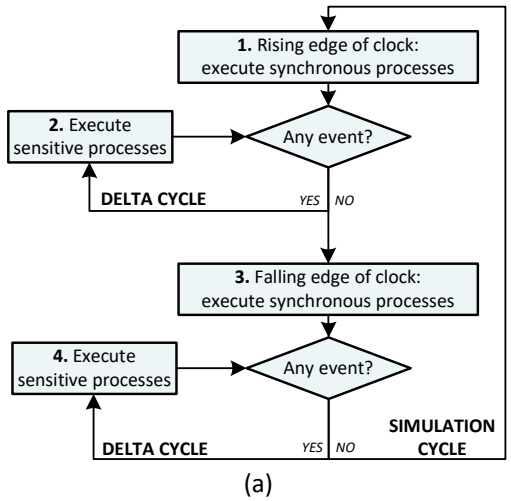
## 5 RTL TO TLM ABSTRACTION OF THE AUGMENTED DIGITAL IP

The recent trend towards the use of abstraction levels higher than RTL has led to the development of methodologies and tools for reusing RTL models of IPs through automatic RTL-to-SystemC TLM abstraction [2, 21]. The constraints detailed in Section 4.1 make de facto the sensors not distinguishable from the starting IP, from the point of view of an automatic abstraction tool. Thus,

any of such tools can in principle be adopted for the abstraction of the augmented IP. The following sections focus on the abstraction process, on the strategies to improve simulation performance, and on the effect of abstraction on the inserted sensors.

### 5.1 Abstraction process

RTL simulation relies on event driven *dynamic simulation kernels*: RTL processes (i.e., concurrent statements) are activated whenever an event in their sensitivity list occurs (e.g., when an input signal changes value). Figure 6.a outlines the execution flow of a typical *HDL simulator kernel*. On the clock rising event, all synchronous processes are run (step 1). Then, if any event has been triggered (e.g., write on a signal), the asynchronous processes sensitive to such an event are woken up (step 2). The routine iteratively goes on until there is no further event. Each of such iterations corresponds to a *delta cycle*, i.e., to a simulation cycle that does not advance simulated time [1] (i.e., the loop around step 2). The same procedure is applied for the falling edge of the clock (steps 3 and 4). When there is no further process to wake up, the simulated time is updated and simulation moves to the next *simulation cycle* (i.e., the arrow going back to step 1). This simulation strategy is common to all HDL languages, despite of the syntactic differences.




```
scheduler(){
  // 1. rising edge of clock
  exec_synchronous_processes();

  // delta cycle
  while(any_event){
    // 2. execute asynchronous processes
    // sensitive to events
    exec_asynchronous_processes();
  }
  // 3. falling edge of clock
  exec_synchronous_processes();

  // delta cycle
  while(any_event){
    // 4. execute asynchronous processes
    // sensitive to events
    exec_asynchronous_processes();
  }
```

(b)

Fig. 6. RTL dynamic scheduling overview (a) and corresponding TLM scheduling code (b).

Despite technical differences, all RTL-to-SystemC TLM abstraction tools generate SystemC TLM code by translating HDL statements into C++ statements (e.g., RTL processes are translated into C++ functions), encapsulated by TLM interfaces and primitives. However, SystemC TLM simulation is almost timeless: simulation proceeds with remote function calls between components, thus bypassing the underlying SystemC RTL scheduler. When moving the RTL IP to TLM, it is thus necessary to preserve the simulation semantics, e.g., in terms of cycle-based accuracy.

Figure 6.b shows how the RTL scheduling is reproduced in TLM. The key ingredient is the `scheduler()` function, that emulates one simulation cycle of the RTL scheduler. The body of the function reproduces step by step the scheduler. First it invokes all synchronous processes, to emulate the rising edge of the clock (step 1). Then, it reproduces the delta cycle, by checking if any value change occurred and executing the corresponding asynchronous processes (step 2). The same is reiterated for emulating the falling edge of the clock (steps 3 and 4). It is important to note that,

in the TLM abstracted version, processes are mapped onto C++ functions, and signals and ports are mapped onto C++ variables. The `scheduler()` function is then wrapped by TLM primitives: each invocation of the TLM primitive corresponds to one simulation cycle of the original RTL design. For more details on the abstraction procedure, please refer to [12, 13].

The key characteristics of such abstraction methodology is that it preserves cycle-based accuracy w.r.t. the starting RTL code: each RTL clock cycle is mapped onto one `scheduler()` call, and the `scheduler()` function reproduces all phases of the RTL scheduler, delta cycles included.

### 5.2 Sensor-aware abstraction

The inserted sensors have different characteristics in terms of timing accuracy. The TLM abstraction must reach a good compromise between timing accuracy and simulation performance, thus two different strategies are defined.

*5.2.1 TLM simulation of the Razor sensor.* To preserve the main characteristics of the Razor sensor, it is necessary to map the RTL cycle-accurate simulation to TLM transitions. Given that the standard abstraction preserves the accuracy to the clock cycle, and that the augmented RTL IP is cycle-accurate w.r.t. one clock signal, no modification shall be applied to the scheduler, that is as in Figure 6.b. This allows to preserve the correct behavior of the sensor and to map each *CLK* period to one TLM transaction, as outlined in Figure 7. This scenario applies to all sensors that preserve accuracy w.r.t. the IP clock cycle, with no finer grain clocks.

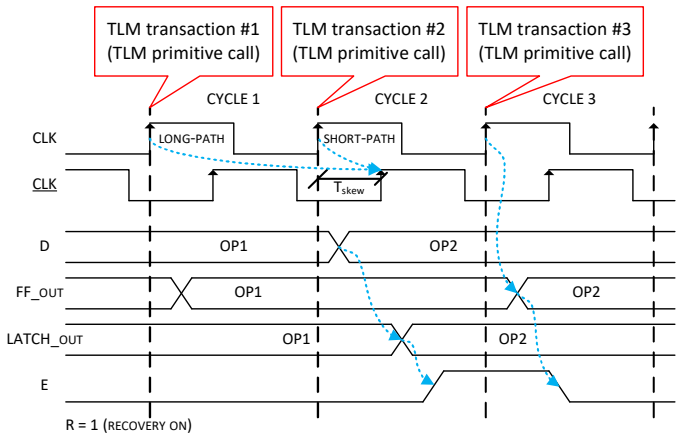



Fig. 7. Mapping of RTL waveforms to TLM transactions for the Razor sensor.

*5.2.2 TLM simulation of the Counter-based sensor.* The Counter-based sensor is sensitive to two clock signals, i.e., the IP clock signal and the high frequency clock. To verify the characteristics of the Counter-based sensor, the TLM code must thus reproduce the presence of both the clock signals, by finding at the same time a good trade off with performance.

The TLM code generated as in Figure 6.b is cycle accurate w.r.t. only one of the clock cycles. Thus, a different strategy is used, as depicted in Figure 8. The main code skeleton is as in Figure 6.b; however, the presence of a higher frequency clock forces some code modifications. To avoid the generation of a high number of TLM transactions, a TLM transaction still corresponds to one cycle of the clock with larger period (`MAIN_CLK`). Vice versa, the second clock signal (`HF_CLK`) is abstracted as a standard signal, so that a number of its cycles are wrapped into one TLM transaction.

This is reflected in the TLM scheduler code (Figure 8.b), where 10 cycles of (HF_CLK) correspond to a single cycle of (MAIN_CLK). This mechanism allows to preserve cycle accuracy also w.r.t. the higher frequency clock, even if one of its cycles does not correspond to an entirely new transaction.

This scenario applies not only to the Counter-based sensor, but also to all sensors that require the presence of a finer grain clock, to perform more precise computation or to precisely measure and monitor metric values.

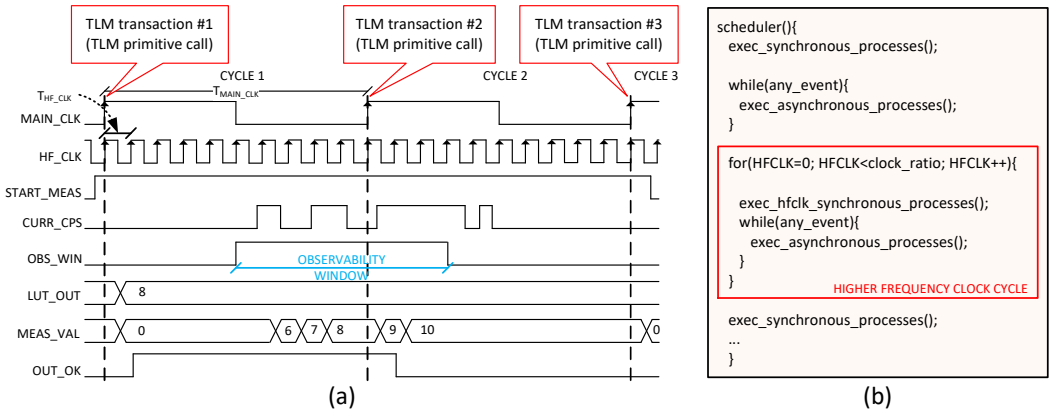



Fig. 8. Mapping of RTL waveforms to TLM transaction sequences for the Counter-based sensor (a) and TLM scheduling code updated with the support for the higher frequency clock (b).

### 5.3 Speeding up TLM simulation: data type abstraction

The standard RTL-to-TLM abstraction process maps the HDL data types to the corresponding SystemC data types, thus preserving data type accuracy and multi-value logic. Nevertheless, data types severely affect SystemC simulation performance with respect to other HDLs [20], and the impact gets even worse at SystemC TLM, where bit accuracy is useless and the slow implementation of data types limits the potential simulation performance of the abstraction paradigm.

To deal with such limitations, the proposed approach adopts HDTLib, an efficient library that provides a faster implementation of all the HDL-oriented data types [11]. HDTLib consists of five data types: a 4-value logic vector class, a 2-value bit vector class, a single logic value class, a signed and an unsigned integer class. In order to achieve a significant performance improvement, HDTLib maps data types on statically allocated arrays of unsigned integers. All operations are implemented on words, instead of single bits, and rely on Karnaugh maps, rather than on lookup tables, to achieve faster access to bitwise truth tables. A complete presentation of HDTLib is available in [10, 11, 51].

Together with such implementation solutions, HDTLib improves simulation speed by considering that bit accurate data types would describe HW characteristics that are useless for the high-level functional verification. At TLM, it is thus possible to accept a loss in terms of accuracy, implicit in the abstraction process, to fasten simulation and verification. To achieve this, multi-valued logic data types (i.e., X, Z, 0, 1) are thus mapped into a two-valued logic type (i.e., 0, 1). The Z and X values are replaced, e.g., with 0, by considering the limitations and effects of this replacement on the actual designs [10, 11].

## 6 MODELING AND INJECTION OF DELAY MUTANTS

Once the RTL IP is abstracted at TLM, RTL delays must be simulated also at TLM level, so to verify the effectiveness of the delay sensors. Given the untimed nature of TLM, it is necessary to model delays as *mutants*, small alterations of the code used to deviate from the expected behaviour (see Section 2.3).

The methodology has to deal with the problem of the loss of accuracy caused by the TLM abstraction process: time accuracy is indeed reduced to an abstract timeline divided into clock cycles and delta cycles, which does not allow for a lower level representation of delays. The proposed *mutation model* solves this problem by preserving, in a more abstract way, delays in the generated TLM models. Such an abstraction of delay is implemented by postponing the signal update (i.e., assignment statement) forward in the simulation time line. This is enabled by the fact that the adopted abstraction process preserves cycle accuracy with respect to the original RTL code. Delayed assignments are then applied in different points of the simulation cycle, thus postponing the assignment and mimicking the effect of a delay.

The different classes of mutants and their application are depicted in Figure 9, that exemplifies mutant injection on a simple example, i.e., the assignment `sig1 = a000` in a synchronous process. Figure 9.e shows the original VHDL process. Figure 9.f is the corresponding C implementation as a function, that is invoked during the rising edge phase of the standard TLM scheduler (the first `exec_synchronous_processes()` in Figure 9.a). To postpone the assignment, the function is broken in two, as shown in Figure 9.g-h: the function invoked during the rising edge phase assigns the value of `a000` to a temporary variable, and this value is passed to variable `sig1` later on, by invoking the function `apply_mutant_sig1()`. The following of this section describes how this modifies the standard TLM scheduling routine.

The standard TLM scheduling routine has only two main synchronization points, i.e., the rising edge and the falling edge of the clock (note that the delta cycles do not have any timing connotation, as they are supposed to happen in zero time). Thus, the first two classes of mutants reproduce:

(1) *Minimum delay mutant.* The actual assignment to the signal is postponed by one delta cycle. In the example of Fig. 9.b, the assignment to `sig1` becomes effective right after the rising edge of the clock (i.e., at the first delta cycle, see Figure 10.a), when function `apply_mutant_sig1()` is invoked by the TLM scheduler. In this way, the delay sensor will receive the updated value of `sig1` after the rising edge of the clock, and thus identify the error.
(2) *Maximum delay mutant.* The assignment to `sig1` is postponed just before the next edge of the clock signal (see Figure 9.c). In the example of Fig. 9.c, the statement is delayed just before the falling edge of the clock (see Figure 10.a). In this way, the delay sensor will receive the updated value of `sig1` with a long delay, and it will thus identify the error.

When the adopted delay sensor introduces a secondary clock, i.e., the high frequency clock, the `scheduler()` function features more synchronization points (Figure 8.b). The high frequency clock requires indeed the introduction of inner loops, that preserve cycle accuracy w.r.t. the high frequency clock. Thus, each main clock period is divided into smaller time units, each corresponding to one period of the high frequency clock. We thus achieve a finer time granularity. This extended notion of time allows to have a finer timing accuracy, and to introduce a third class of mutants:

(3) *Delta delay mutant.* The actual assignment to the signal is postponed exactly by a number of high frequency clock cycles. In the example of Fig. 9.d, the statement is delayed of three `HF_CLK` clock cycles, thus introducing a delay between two and three periods of the high frequency clock. This allows to create a finer correspondence between RTL delays and TLM delays, and to have a (gross) estimate of the delay being simulated at TLM, measured in terms

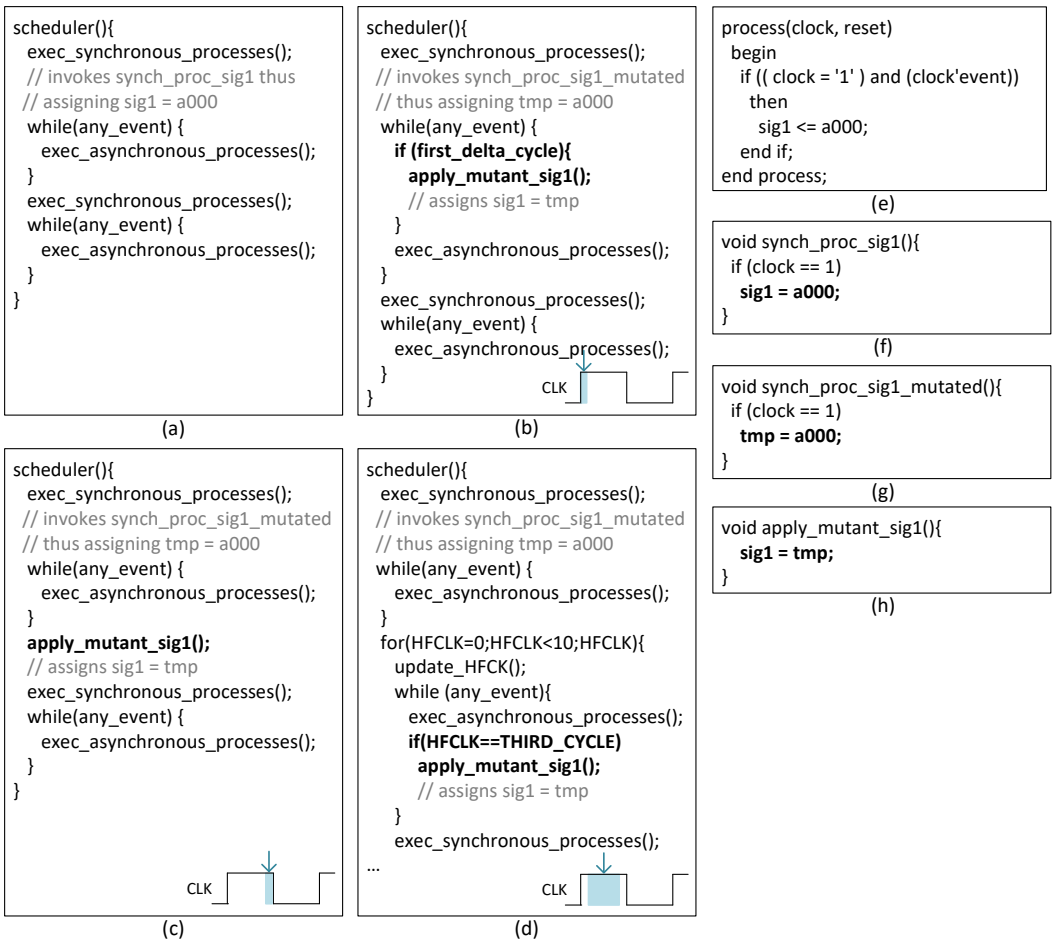



Fig. 9. The proposed mutants for the TLM model: high-level representation of the TLM scheduling without mutants (a), a *minimum delay* mutant example (b), a *maximum delay* example (c), and an example of delta delay mutant (d). The snapshots of code apply delays to the assignment `sig1 = a000`, for which the figure shows the original VHDL code (e), the corresponding C function (f), and the effect of mutant injection (g-h).

of HF_CLK cycles (as depicted in Figure 10.b), as opposed to maximum and minimum delays, that give no indication (Figure 10.a).

### 6.1 Verification of digital IPs augmented with Razor sensors

The working time window of the Razor sensor spans from the rising edge of the clock to the falling edge of the clock.

To detect delays in this time window, both the *Minimum* and *Maximum delay* mutants are adopted. The minimum delay allows to detect signals with the minimum possible delay, i.e., delayed of one delta cycle. The maximum delay mutant allows to model signals with a delay of half of *CLK* cycle, as explained in Section 4.1.1. This covers the two extremes of the time window, as shown in Figure 10. Thus, mutants allow to reproduce delays in the range that must be covered by the Razor sensor, and thus to validate the effectiveness of the Razor sensor at detecting and correcting delays.

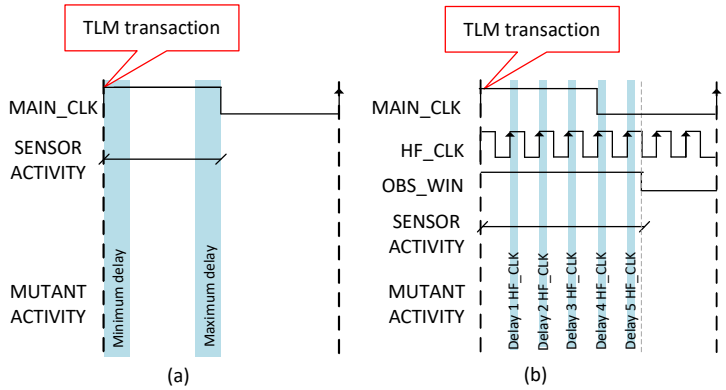



Fig. 10. Application of mutant injection for the Razor sensor (a) and the Counter-based sensor (b)

### 6.2 Verification of digital IPs augmented with Counter-based sensors

The main difference between the Razor sensor and the Counter-based sensor is that the latter provides an estimation of the delay of signal propagation. This reduces the effectiveness of the Minimum and Maximum delay mutants, that introduce delays of one delta cycle or of half of a clock cycle. Here comes the need for the third type of mutant, the *Delta mutant*.

The Delta delay mutant is applied to insert a given number of high frequency clock cycles (*HF_CLK*) of delay on a target signal, as shown in Figure 10. The Delta delay mutant allows the detection feature of the Counter-based sensor to be verified at TLM for the whole observability window (*OBS_WIN*), by preserving the characteristics of maximum resolution (*HF_CLK* period), maximum error (half of *HF_CLK* cycle) and dynamic range provided by the RTL simulation, as explained in Section 4.1.2.

## 7 MUTATION ANALYSIS OF THE AUGMENTED DIGITAL IPS WITH THE PROPOSED MUTANTS

The last step of the proposed flow is the simulation of the TLM IP injected with mutants, with the goal of verifying the behavior of the inserted delay monitors.

To achieve this result, the TLM IP injected with delay mutants is simulated in parallel with a non-injected TLM IP, with the same input stimuli. Inputs are typically provided by the testbench shipped with the IP, written at design time to stress and check all the features of the RTL IP and to dynamically verify its correctness [6]. At this point of the design flow, the testbench must achieve good code coverage i.e., it should execute all (or most of) paths, to ensure sound functional verification [7], and it is thus reasonable to assume that the testbench stimulates the identified critical paths. To allow mutant detection, the testbench must lead to a change of the value given in input to the delay sensor (i.e., from 0 to 1 or vice versa), so that the delayed assignment becomes visible. If a testbench is not available, or in case the testbench is not complete, stimuli can be generated by applying any stimuli generation technique available at the state of the art [25, 34]. Note that discussing this point is out of the scope of this paper.

In order to verify the behavior of all inserted sensors, simulation is run once for each sensor, by activating the corresponding delay mutant. This introduces the desired delay on the critical path of the sensor under evaluation. Throughout the simulation, outputs are analyzed by comparing the results of the two TLM IPs: if the outputs differ, then the mutant has been killed and the delay

produced effects on the functionality of the IP. Once that this condition is met, it is possible to verify the behavior of the inserted sensor by observing its outputs.

In case of a digital IP augmented with the Razor sensor, one must observe the output port $E$ of the Razor, as it is used to notify that the FF and the shadow latch contain different values. Thus, the $E$ port is observed in combination with all the output ports of the IP:

- If $E = 1$, the Razor sensor detected the injected delay, and the correction feature of the Razor (i.e., correction of output values with some clock cycles of delay) can be observed on the output ports of the IP.
- If $E = 0$ (and the mutant is switched on), either the mutant has not been activated because the testbench has failed to generate a proper input sequence to stress the mutant (i.e., either it did not reach the assignment statement or the assignment did not change the value of the signal), or the mutant models a delay outside the range of detection of the sensor. In the former case, the IP outputs of the injected version match with those of the non injected one. In the latter case, the IP outputs of the injected vs. non inject versions do not match, and the sensor failed at verifying the delay on the critical path.

In case of a digital IP augmented with a Counter-based monitor, one must observe the $MEAS\_VAL$ port of the Counter-based sensor, as it contains the estimated delay:

- If $MEAS\_VAL \neq 0$, the corresponding mutant has been activated and detected;
- If $MEAS\_VAL = 0$ (and the mutant is switched on), the testbench has failed to generate a proper input sequence to stress the mutant.

## 8 EXPERIMENTAL RESULTS

This Section proves the effectiveness of the proposed design paradigm and verification methodology on three complex case studies. All experiments have been run on a 64-bit server with 8 3.40 GHz cores and 16GB of RAM, and running Kubuntu 14.04 Linux OS. Simulation times are calculated as an average over a number of executions.

### 8.1 Case studies

The proposed cross-level verification methodology has been applied to three case studies:

- an open-source implementation of the MIPS R3000A microprocessor, supporting the MIPS I Instruction Set Architecture [40];
- the digital sub-system of a custom DSP for heart rate detection, that applies digital filters and integrators to blood flow imaging [29];
- a digital decimation filter contained in a MEMS smart microphone systems, developed with Matlab HDL Coder.

Table 1 shows the main characteristics of the IPs. The table reports the number of lines of code of the starting RTL (Column *RTL (loc)*), the number of primary input and output pins (Columns *PI (#)* and *PO (#)*), the number of flip-flops (Column *FF (#)*) and the area in terms of equivalent NAND2 gates (Column *Gates (#)*) of each IP. Moveover, column *Monitored paths (#)* shows the number of identified critical paths that require the insertion of delay sensors, while column *Processes* reports the number of synchronous and asynchronous processes. Timing and area results refer to a synthesis performed on 45nm CMOS technology from STM, and to the voltage-frequency points reported in Columns $V_{DD}$ *[V]* and $f_{clk}$ *[GHz]*.

Table 1. Characteristics of the IPs used as case studies

| Digital IP | RTL (loc) | PI (#) | PO (#) | $V_{DD}$ [V] | $f_{clk}$ [GHz] | FF (#) | Gates (#) | Processes | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Synch. | Asynch. |
| Plasma | 1,893 | 36 | 132 | 1.05 | 0.2 | 1297 | 14286 | 7 | 94 |
| DSP | 1,274 | 22 | 22 | 1.05 | 2 | 536 | 8098 | 2 | 67 |
| Filter | 508 | 5 | 15 | 1.05 | 1 | 128 | 2255 | 11 | 34 |

### 8.2 Insertion of delay monitors

The analysis described in Section 4.2 has been applied by using Synopsys PrimeTime v2016.06 [47]. The number of identified critical paths for each design is reported in Table 2 (Column *Critical paths (#)*). The endpoint of each critical path identifies the location where a delay sensor must be inserted. Thus, the number of inserted sensors is the same as the number of critical paths (Column *Sensors (#)*). By using the strategy in Section 4.2, we generated two versions of each IP, one using Razor sensors and the other using Counter-based sensors. Column *RTL (loc)* shows that the number of lines of code is now larger than the original implementation, and that the Counter-based versions required more lines of code, due their more complex implementation and to the presence of the secondary clock.

Table 2 also reports the time required by the Static Timing Analysis phase required to identify sensors locations and of the following automated RTL modification (column *STA time (s)*). This analysis and manipulation step required less than 10s for all designs. It is important to note that this process is mandatory also when performing verification at RTL level, as it is necessary to identify the paths to be monitored. Thus, it does not constitute an overhead to the verification flow.

Table 2. Characteristics of the insertion of delay monitors

| Digital IP | STA time (s) | Critical paths (#) | Sensors | | RTL (loc) |
|---|---|---|---|---|---|
| | | | Type | Inserted (#) | |
| Plasma | 9.45 | 29 | Razor | 29 | 2,308 |
| | | | Counter | 29 | 2,844 |
| DSP | 8.51 | 34 | Razor | 34 | 3,025 |
| | | | Counter | 34 | 14,959 |
| Filter | 8.22 | 24 | Razor | 24 | 1,008 |
| | | | Counter | 24 | 6,178 |

### 8.3 RTL-to-TLM abstraction

Table 3 reports information on the abstracted SystemC TLM descriptions in terms of lines of code and of simulation time. RTL-to-TLM abstraction (Column *Abstracted TLM*) increases the number of lines of code, as the abstraction process introduces the TLM primitives and it requires to explicitly model the scheduling routine, to decouple functionality from the HDL RTL scheduler. Nonetheless, TLM code generation was almost instantaneous and faster than 1 minute for all versions of all IPs. The generated code is on average 3.05 times faster than the original RTL implementation, which was simulated using a state-of-the art industrial tool, i.e., Mentor Graphics QuestaSim, version 10.6. It is possible to note that the speedup is on average larger for the Counter-based versions (3.2x), given the more complex implementation of the Counter-based monitor and its high frequency clock, thus leaving more space to optimization.

Table 3. Characteristics and simulation performance of the generated TLM code.

| Digital IP | Delay sensors | RTL Time (s) | Abstracted TLM (loc) | Abstracted TLM Time (s) | Abstracted TLM Speedup w.r.t. RTL |
|---|---|---|---|---|---|
| Plasma | Razor | 146.69 | 9,314 | 56.42 | 2.60x |
| | Counter | 295.94 | 14,765 | 106.45 | 2.78x |
| DSP | Razor | 227.98 | 8,306 | 71.02 | 3.21x |
| | Counter | 1,547.27 | 56,384 | 520.97 | 2.97x |
| Filter | Razor | 301.65 | 4,617 | 101.23 | 2.98x |
| | Counter | 1,314.71 | 35,277 | 345.98 | 3.80x |

The application of data type optimization (Column *Optimized TLM*) allowed to further improve TLM simulation time, as reported in Table 4. The generated code is on average 1.34x faster, thanks to the adoption of HDTLib rather than standard SystemC data types. The lighter implementation of the data type library thus positively impacts on the performance with respect to the original RTL implementation, reaching an average speedup of 4.03x.

Table 4. Characteristics and simulation performance of the generated optimized TLM code.

| Digital IP | Delay sensors | Optimized TLM Time (s) | Optimized TLM Speedup w.r.t. TLM | Optimized TLM Speedup w.r.t. RTL |
|---|---|---|---|---|
| Plasma | Razor | 33.99 | 1.66x | 4.32x |
| | Counter | 62.99 | 1.69x | 4.70x |
| DSP | Razor | 65.16 | 1.09x | 3.50x |
| | Counter | 434.14 | 1.20x | 3.56x |
| Filter | Razor | 77.87 | 1.30x | 3.87x |
| | Counter | 311.69 | 1.11x | 4.22x |

### 8.4 Injection of delay mutants and mutation analysis

The mutants defined in Section 6 have been injected in the SystemC TLM models through ADAM, the *Automatic Delay Analysis and Mutation* tool, implemented on top of the HIFSuite API [21]. The tool takes in input the names of the RTL signals connected to the delay monitors and the kind of mutant to inject, and it automatically applies the necessary code modifications.

The number of injected mutants is reported in Table 5 (Column *Mutants (#)*). The versions containing Razor sensors are injected with minimum delay mutants and maximum delay mutants, to test that both conditions are detected by the sensors. Vice versa, Counter based sensors introduce also an intermediate granularity of time, thanks to the higher frequency clock. Thus, the versions with Counter based mutants are injected also with delta delay mutants. Mutant injection increased the number of lines of code for each code version, due to the insertion of mutants and to the management processes (Column *Injected TLM (loc)*). This impacts simulation time, that is increased by an average of 43%. However, it is important to note that the injected version is still 2.83x faster than the original RTL implementation on average, thus allowing effective verification of the inserted delay monitors.

Table 5. Characteristics and results of the application of mutation analysis.

| Digital IP | Delay sensors | Injected TLM | | | Mutants | | | Errors risen (%) |
|---|---|---|---|---|---|---|---|---|
| | | (loc) | Time (s) | Speedup w.r.t. RTL | (#) | killed (%) | corrected (%) | |
| Plasma | Razor | 13,292 | 45.00 | 3.26x | 58 | 100.0 | 100.0 | 100.0 |
| | Counter | 18,743 | 88.08 | 3.36x | 87 | 100.0 | n.a. | 66.7 |
| DSP | Razor | 39,297 | 98.27 | 2.32x | 68 | 100.0 | 100.0 | 100.0 |
| | Counter | 87,375 | 661.23 | 2.34x | 102 | 100.0 | n.a. | 88.4 |
| Filter | Razor | 17,833 | 105.11 | 2.87x | 48 | 100.0 | 100.0 | 100.0 |
| | Counter | 48,494 | 446.21 | 2.82x | 72 | 100.0 | n.a. | 50.1 |

### 8.5 Mutation analysis

The mutation analysis presented in Section 7 has been finally applied to the obtained TLM models. To generate the input stimuli, we relied on the RTL testbenches of the IPs, that have been applied both to the abstracted TLM version and to the TLM injected with delay mutants. Applying mutation analysis required to simulate the TLM versions once per inserted sensor: this further increases the effectiveness of the fast TLM simulation, that is still on average 4.03x faster than RTL simulation.

Results of the application of mutation analysis are reported in Table 5 (Column *Mutation analysis*). Mutation analysis allowed us to kill all mutants, i.e., to detect all injected delays in both versions of each IP. Additionally, the Razor versions notified and corrected all the injected delays (columns *Error risen (%)* and *Mutants corrected (%)*, respectively). To validate these results, we simulated the same scenario at RTL, by injecting delays through explicitly delayed assignments (e.g., by using `after` constructs). RTL simulation required much longer simulation times (Column *RTL Time (s)* of Table 3), but the percentages of detected and corrected delays, and of risen errors are identical (100% for all designs).

The Counter-based sensor notifies as errors only delays that meet a given threshold (see Section 4.1.2). In our experimental results, the threshold has been set, in a monitor look-up table, equal to 8 periods of the high frequency clock. Mutants modelling delays below such a threshold have been detected, but they have not been notified as errors, as delays are considered tolerable by the system, i.e., the system is robust enough to handle them. To validate the correctness of the identified errors, we reproduced the same scenario at RTL. We indeed considered that TLM preserves cycle accuracy with respect to the high frequency clock (Figure 8). This can be used to have a (gross) estimate of the simulated delay, depending on when the assignment takes place with respect to the high frequency clock. It was thus possible to model delays at RTL and TLM that fall within the same period of the high frequency clock. Given that the Counter-based sensor measures delay in terms of high frequency clock periods, such delays are de facto identical for the sensor. As a result, the number of errors risen at RTL and at TLM was identical, thus validating the effectiveness of TLM simulation w.r.t. RTL simulation of delay sensors.

## 9 CONCLUSIONS

This paper presented a methodology to verify at system-level (TLM) digital IPs augmented with embedded timing monitors. With the abstraction to TLM, the delay on critical paths is detected since the functionality of the implemented timing monitors is preserved. This allowed us to catch at TLM the side-effects of many physical properties on the performance of digital IPs. The methodology has been applied to three case studies, which have been augmented with two types of sensors, to show that both precise monitoring and correction of errors are possible. The results show that

the proposed mutants effectively reproduce the effect of delays at TLM and that an accurate and efficient verification of the delay sensors is possible at levels of abstractions higher than RTL, with a substantial improvement of verification speed. Additionally, all features of the proposed sensors have been preserved, from delay identification to delay correction, depending on the starting monitor characteristics.

## REFERENCES


[1] Accellera. 2006. IEEE Standard SystemC Language Reference Manual. http:///ieeexplore.ieee.org. (2006).

[2] M.A. Alam, H. Kufluoglu, D. Varghese, and S. Mahapatra. 2007. Carbon Model Studio. *Microelectronics Reliability* 47, 6 (2007), 853 – 862. http://carbondesignsystems.com/.

[3] M.A. Alam, H. Kufluoglu, D. Varghese, and S. Mahapatra. 2007. A comprehensive model for PMOS NBTI degradation: Recent progress. *Microelectronics Reliability* 47, 6 (2007), 853 – 862.

[4] J. R. Armstrong, F. S. Lam, and P. C. Ward. 1992. *Test Generation and Fault Simulation for Behavioral Models.* Prentice Hall.

[5] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, and P.-J. Gil. 2008. Enhancement of Fault Injection Techniques Based on the Modification of VHDL Code. *IEEE VLSI* 16, 6 (2008), 693–706.

[6] J. Bergeron. 2003. *Writing Testbenches: Functional Verification of HDL Models.* Kluwer Academic Publishers, Norwell Massachusetts.

[7] Janick Bergeron. 2003. *Writing Testbenches: Functional Verification of HDL Models.* Springer.

[8] N. Bombieri, D. Drogoudis, G. Gangemi, R. Gillon, E. Macii, M. Poncino, S. Rinaudo, F. Stefanni, D. Trachanis, and M. van Helvoort. 2013. SMAC: Smart Systems Co-Design. In *Proc. of Euromicro DSD*. 1–7.

[9] N. Bombieri, F. Fummi, V. Guarnieri, and G. Pravadelli. 2013. Testbench Qualification of SystemC TLM Protocols through Mutation Analysis. *IEEE TCOMP* PP, 99 (2013), 1–14.

[10] N. Bombieri, F. Fummi, V. Guarnieri, F. Stefanni, and S. Vinco. 2011. Efficient implementation and abstraction of systemc data types for fast simulation. In *Proc. of IEEE/ECSI FDL*. 1–7.

[11] N. Bombieri, F. Fummi, V. Guarnieri, F. Stefanni, and S. Vinco. 2012. HDTLib: an efficient implementation of SystemC data types for fast simulation at different abstraction levels. *Design Automation for Embedded Systems* 16, 2 (2012), 115–135.

[12] N. Bombieri, F. Fummi, and G. Pravadelli. 2011. Automatic Abstraction of RTL IPs into Equivalent TLM Descriptions. *IEEE Trans. Comput.* 60, 12 (2011), 1730–1743.

[13] N. Bombieri, F. Fummi, and S. Vinco. 2015. A Methodology to Recover RTL IP Functionality for Automatic Generation of SW Applications. *ACM TODAES* 20, 3, Article 36 (2015), 26 pages.

[14] S. Borkar. 2005. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE MICRO* 25, 6 (2005), 10–16.

[15] K.A. Bowman, J.W. Tschanz, Nam Sung Kim, J.C. Lee, C.B. Wilkerson, S.L. Lu, T. Karnik, and V.K. De. 2009. Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance. *IEEE JSSC* 44, 1 (2009), 49–63.

[16] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D Blaauw, T. Austin, K. Flautner, and T. Mudge. 2006. A self-tuning DVS processor using delay-error detection and correction. *IEEE JSSC* 41, 4 (2006), 792–804.

[17] S. Das, C. Tokunaga, S. Pant, Wei-Hsiang Ma, S. Kalaiselvan, K. Lai, D.M. Bull, and D.T. Blaauw. 2009. RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance. *IEEE JSSC* 44, 1 (2009), 32–48.

[18] R. Datta, G. Carpenter, K. Nowka, and J.A. Abraham. 2006. A scheme for on-chip timing characterization. In *Proc. of 24th IEEE VLSI Test Symp.* 24–29.

[19] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer* 11, 4 (April 1978), 34–41.

[20] W. Ecker, V. Esen, L. Schonberg, T. Steininger, M. Velten, and M. Hull. 2007. Impact of description language, abstraction layer, and value representation on simulation performance. In *Proc. of ACM/IEEE DATE*. 1–6.

[21] EDALab s.r.l. [n. d.]. HIFSuite. http://http://www.hifsuite.com/. ([n. d.]).

[22] F. Fummi, M. Lora, F. Stefanni, and Vinco S. 2015. *Code Generation Alternatives to Reduce Heterogeneous Embedded Systems to Homogeneity.* Vol. 311. Springer, 103–124.

[23] F. Fummi, M. Lora, F. Stefanni, D. Trachanis, J. Vanhese, and S. Vinco. 2014. Moving from co-simulation to simulation for effective smart systems design. In *IEEE DATE*. 1–4.

[24] S. Ghosh, S. Bhunia, A. Raychowdhury, and K. Roy. 2006. A Novel Delay Fault Testing Methodology Using Low-Overhead Built-In Delay Sensor. *IEEE CAD* 25, 12 (2006), 2934–2943.

[25] V. Guarnieri, N. Bombieri, G. Pravadelli, F. Fummi, H. Hantson, J. Raik, M. Jenihhin, and R. Ubar. 2011. Mutation analysis for SystemC designs at TLM. In *Proc. of IEEE LATW*. 1–6.

[26] V. Guarnieri, M. Petricca, A. Sassone, S. Vinco, N. Bombieri, F. Fummi, E. Macii, and M. Poncino. 2014. A cross-level verification methodology for digital IPs augmented with embedded timing monitors. In *Proc. of ACM/IEEE DATE*. 1–6.
[27] R. Guderlei, R. Just, and C. Schneckenburger. 2008. Benchmarking Testing Strategies with Tools from Mutation Analysis. In *IEEE ICSTW*. 360–364.
[28] N. Hatami, R. Baranowski, P. Prinetto, and H. Wunderlich. 2012. Efficient system-level aging prediction. In *Proc. of IEEE ETS*. 1–6.
[29] D. He, H. C. Nguyen, B. R. Hayes-Gill, Y. Zhu, J. A. Crowe, G. F. Clough, C. A. Gill, and S. P. Morgan. 2012. 64ĂŰ64 pixel smart sensor array for laser Doppler blood flow imaging. 37 (2012), 3060–3062. Issue 15.
[30] Z. He, T. Lv, H. Li, and X. Li. 2013. Test Path Selection for Capturing Delay Failures Under Statistical Timing Model. *IEEE TVLSI* 21, 7 (2013), 1210–1219.
[31] D. Hyunsook and G. Rothermel. 2006. On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques. *IEEE SE* 32, 9 (2006), 733–752.
[32] H.M. Le, D. Grosse, and R. Drechsler. 2012. Automatic TLM Fault Localization for SystemC. *IEEE TCAD* 31, 8 (2012), 1249–1262.
[33] R. Leveugle, D. Cimonnet, and A. Ammari. 2004. System-level dependability analysis with RT-level fault injection accuracy. In *Proc. of IEEE DFT*. 451–458.
[34] P. Lisherness and K. Cheng. 2012. Improving validation coverage metrics to account for limited observability. In *Proc. of IEEE ASPDAC*. 292–297.
[35] Peter Lisherness and Kwang-Ting Cheng. 2010. SCEMIT: A SystemC error and mutation injection tool. In *Proc. of ACM/IEEE DAC*. 228 –233.
[36] A.J. Offutt and R.H. Untch. 2001. Mutation Testing for the New Century. Kluwer Academic Publishers, Chapter Mutation 2000: Uniting the Orthogonal, 34–44.
[37] P.G. Paulin, C. Pilkington, and E. Bensoudane. 2002. StepNP: a system-level exploration platform for network processors. *IEEE DTC* 19, 6 (2002), 17–26.
[38] Songwei Pei, Huawei Li, and Xiaowei Li. 2012. A High-Precision On-Chip Path Delay Measurement Architecture. *IEEE VLSI* 20, 9 (2012), 1565–1577.
[39] G. Pravadelli, D. Quaglia, S. Vinco, and F. Fummi. 2017. Handbook of Hardware/Software Codesign. Springer Science+Business Media Dordrecht, Chapter Semiformal Assertion-Based Verification of Hardware/Software Systems in a Model-Driven Design Framework, 683–720.
[40] S. Rhoads. 2001. Plasma CPU Core. (2001). opencores.org.
[41] M. Rimen, J. Ohlsson, E. Jenn, J. Arlat, and J. Karlsson. 1994. Fault Injection into VHDL Models: The MEFISTO Tool. In *Proc. of IEEE FTCS*. 66–75.
[42] T. Sato and Y. Kunitake. 2007. A Simple Flip-Flop Circuit for Typical-Case Designs for DFM. In *Proc. of IEEE ISQED*. 539–544.
[43] L. Scheffer, L. Lavagno, and G. Martin. 2010. *EDA for IC System Design, Verification, and Testing*. Taylor & Francis.
[44] Alper Sen and Magdy S. Abadir. 2010. Coverage metrics for verification of concurrent SystemC designs using mutation testing. In *Proc. of IEEE HLDVT*. 75–81.
[45] K. Shim, W. Kim, K.-H. Cho, and B. Min. 2012. System-level simulation acceleration for architectural performance analysis using hybrid virtual platform system. In *Proc. of IEEE ISOCC*. 402–404.
[46] Ahish Mysore Somashekar, Spyros Tragoudas, Rathish Jayabharathi, and Sreenivas Gangadhar. 2016. Non-enumerative Generation of Path Delay Distributions and Its Application to Critical Path Selection. *ACM Trans. Des. Autom. Electron. Syst.* 22, 1 (2016), 17:1–17:21.
[47] Synopsys. [n. d.]. PrimeTime Static Timing Analysis. https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html. ([n. d.]).
[48] Ming-Chien Tsai, Ching-Hwa Cheng, and Chiou-Mao Yang. 2008. An All-Digital High-Precision Built-In Delay Time Measurement Circuit. In *Proc. of IEEE VTS*. 249–254.
[49] O.S. Unsal, J.W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. 2006. Impact of Parameter Variations on Circuits and Microarchitecture. *IEEE MICRO* 26, 6 (2006), 30–39.
[50] S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino. 2017. A Layered Methodology for the Simulation of Extra-Functional Properties in Smart Systems. *IEEE TCAD* 36, 10 (2017), 1702–1715.
[51] S. Vinco, V. Guarnieri, and F. Fummi. 2016. Code Manipulation for Virtual Platform Integration. *IEEE TCOMP* 65, 9 (2016), 2694–2708.
[52] Xiaoxiao Wang, M Tehranipoor, S. George, D. Tran, and L. Winemberg. 2012. Design and Analysis of a Delay Sensor Applicable to Process/Environmental Variations and Aging Measurements. *IEEE VLSI* 20, 8 (2012), 1405–1418.