Analog Layer Extensions for Analog/Mixed-Signal Assertion Languages

Dogan Ulus
Dept. of Elecrical & Electronics Eng.
Bogazici University
Istanbul, Turkey
Email: dogan.ulus@boun.edu.tr

Alper Sen
Dept. of Computer Eng.
Bogazici University
Istanbul, Turkey
Email: alper.sen@boun.edu.tr

Faik Baskaya
Dept. of Elecrical & Electronics Eng.
Bogazici University
Istanbul, Turkey
Email: faik.baskaya@boun.edu.tr

***Abstract*—Assertion-based methodology is gaining popularity in analog and mixed-signal (AMS) verification. Early AMS assertion languages are built on digital assertion languages. This results in limited native support to express most low-level aspects of AMS properties. We present three analog layer extensions to increase analog expressiveness in AMS assertion languages. We first describe the concept of haloes, an implicit way to handle tolerance values of analog signals in assertions. Then, booleanization of analog signals using dual-threshold is introduced to solve problems caused by fluctuations on signals. Finally, we integrate analog measurement operators into assertions. We validate our extensions using our prototype tool on a 10-bit two-stage pipelined analog-to-digital converter design.**

I. Introduction

Assertion-based verification (ABV) has gained a great deal of popularity in the past decade, and it has been widely adopted by digital design community thanks to its practicality and simplicity. In an ABV flow, assumptions of the designers and the design intent are captured by assertion statements. These statements can monitor a design to ensure correct behavior of individual simulation runs, and report if any unintended situation happens. This allows us to detect bugs at their source, and assertions improve design observability and controllability although they do not ensure 100% design correctness.

Applying the ABV methodology to Analog and Mixed-Signal (AMS) designs can bring the same benefits that the digital design community has enjoyed. As the number and complexity of AMS blocks are increasing in System-on-Chip (SoC) devices, traditional AMS verification is not sufficient to satisfy today's highly-ambitious time-to-market and first-pass-silicon requirements. Assertions for AMS designs can formalize the specification of the design, and automate the evaluation of simulation results. Such a structured methodology reduces manual effort, increases reusability and ultimately leads to a productivity increase in AMS verification.

Assertion-based verification for AMS designs borrows a lot from digital verification. Using predicates, analog quantities are converted into Boolean signals, and temporal logic operators can then check time-domain specifications on these Boolean signals. Although this is an effective methodology, it does not take some important analog facts into account as well as it lacks some useful analog constructs and routines. Integrating these constructs and routines into AMS assertions can provide a more natural expression of analog properties and increase the quality of AMS verification.

In this paper, we focus on analog expressiveness of AMS assertion languages. By taking the analog nature of things into account, we extend the *analog layer* of AMS assertion languages with three useful extensions. We propose the following extensions.

- *Haloes*: An intuitive method to express tolerances of analog signals implicitly in assertions.
- *Dual-threshold Booleanization*: A natural way to convert analog signals into Boolean signals without glitches.
- *Analog measurements*: Common numerical routines to check analog signal properties.

II. Related Work

A number of formal approaches for analog circuit verification is presented in [1]. Several works in the literature use these formal methods and they have formalized simulation-based approaches to verify analog circuits. Analog Specification Language (ASL) in [2], [3] was proposed to define analog properties for state-space analysis. System of Recurrence Equations (SRE) in [4] and Petri-nets in [5] were used to verify analog circuits in a formal way. On the other hand, scalable but incomplete simulation-based formal approaches, which don't suffer from state-space explosions, have been studied in [6], [7], [8].

In [6], two verification approaches, an event-checker library based AMS-OVL and an assertion based AMS-SVA, were presented for Verilog-AMS designs. Similarly, in [7], Lammermann et al. presented a general assertion-based methodology, and an online temporal checker library for SystemC-AMS designs. Their specification language Mixed-Signal Assertion Language (MSAL) is implemented for discrete-time, and the synchronization between MSAL layers is performed at every clock cycle. It includes specific analog operators providing mappings from analog sampling time to digital clock.

In [8], the *Signal Temporal Logic* (STL), which extends the *Metric Interval Temporal Logic* (MITL) [9], was presented to monitor time-domain properties of continuous signals. Analog layer of STL incorporates comparison operators to booleanize analog signals and a few useful operators like distance, rise and fall.

In our previous work [10], we started to extend AMS assertion languages, introduced the halo concept and provided two case studies. The current paper is another step to improve analog expresiveness by extending AMS assertion languages with useful constructs.

III. AMS Assertion Languages

Early AMS assertion languages are usually based on *Linear Temporal Logic* (LTL) [11], and extend its expressiveness towards real-valued signals in continuous time-domain. These languages consist of several abstraction layers. They are influenced by *Property Specification Language* (PSL) [12] where layers are hierarchically placed. Besides PSL layers such as *Boolean* and *Temporal layers*, AMS assertion languages incorporate a new layer to define and check analog properties. This *Analog layer* extracts information from analog signals, and converts them into Boolean signals. For example, a designer may want to check if the voltage value of an analog signal is always less than 1.8V. The less-than operator in *Analog layer* returns

978-1-4799-0524-9/13/$31.00 ©2013 IEEE

TABLE I
OUR AMS ASSERTION LANGUAGE GRAMMAR

Temporal Layer	$TempExpr$	::=	$\circledcirc BoolExpr$
		|	$\circledcirc TempExpr$
		|	$TempExpr \bullet TempExpr$
		|	$\neg TempExpr$
Boolean Layer	$BoolExpr$	::=	$HaloExpr \boxdot HaloExpr$
		|	$HaloExpr \boxtimes HaloExpr$
		|	$MeasExpr \boxtimes Const$
		|	$BoolExpr \bullet BoolExpr$
		|	$\neg BoolExpr$
Analog Layer	$HaloExpr$	::=	$\mathcal{H}(SignalExpr, \text{params})$
		|	$SignalExpr$
	$MeasExpr$	::=	$\mathcal{M}(SignalExpr, \text{params})$
	$SignalExpr$	::=	$AnalogSignal$
		|	$SignalExpr \odot SignalExpr$
		|	$Const$

Operator Types	Symbols
Temporal	$\circledcirc \in \{F, G, O, H\}$
Binary Boolean	$\bullet \in \{\wedge, \vee, \rightarrow\}$
Binary Negation	$\neg$
Halo Comparison	$\boxdot \in \{\text{SGT}, \text{NGT}, \text{CVR}, \text{CVD}, \text{NLT}, \text{SLT}\}$
Relational	$\boxtimes \in \{>, <, \geq, \leq\}$
Halo Calculation	$\mathcal{H}$
Measurement	$\mathcal{M}$
Arithmetic	$\odot \in \{+, -, *, /\}$

a Boolean signal that shows the regions were the condition is satisfied in time domain. After analog signals are booleanized, and converted into Boolean signals, boolean and temporal operators can be applied to check whether they satisfy desired properties or not.

In Table I we show the grammar of our AMS assertion language. The temporal and boolean operators have their usual semantics. Specifically, G, F are bounded future temporal logic operators denoting $always$, and $eventually$, whereas O and H are bounded past temporal logic operators denoting $once$, and $historically$. The grayed out region in Table I shows our extensions, namely halo calculation, Booleanization, and measurement operators, which we explain in the following sections.

An example property using our grammar is as follows:

$$G(\textit{TRIG} \rightarrow F_{[0.0:0.5]}(\mathcal{H}(S) \text{ CVR } \mathcal{H}(lvl)))$$

where *TRIG*, *S* and *lvl* denote a Boolean signal, an analog signal and a constant value, respectively. This property always checks that if the trigger signal $TRIG$ is true, then the halo of *S* covers the halo of *lvl* in 0.5 time units.

IV. HALOES

Analog signals are usually associated with tolerance values. One reason for this is that these signals are subject to random and systematic fluctuations in reality. Besides noise sources in analog circuits, some analog facts like charge injection can contribute to the fluctuation of numerical simulation values. Another reason is that specifications themselves can impose a region of tolerated values around a nominal value or a reference analog signal. In fact, comparison with reference signals is a common case in analog design. For example, outputs of a high-level model can be compared with the outputs of a transistor-level model or pre-layout simulation results can be compared with post-layout simulation results.

We introduced the halo concept for analog signals in [10] to handle tolerance values in the above mentioned situations. Given an analog signal, we define the $halo$ as a pair of boundary signals around the analog signal. We consider the area between the boundary signals as acceptable values for that analog signal. Besides being able to handle tolerance values in AMS assertions, using haloes also provides more options to define what is acceptable for an analog signal. We proposed four different methods to calculate haloes in [10]. These methods involve user-defined constants, signal processing, and statistical methods.

We can have six possible relations between two haloes, namely *strictly-greater-than* (SGT), *nearly-greater-than* (NGT), *covers* (CVR), *is-covered* (CVD), *nearly-less-than* (NLT), and *strictly-less-than* (SLT). These relations are visualized in Figure 1.

V. DUAL-THRESHOLD BOOLEANIZATION

We define analog (real-valued or continuous) signals as functions from the time-domain $\mathbb{T} = \mathbb{R}^+$ to a state-space $\mathbb{X} \subseteq \mathbb{R}^n$, where n is the number of predicates, as in [13]. The *Booleanization* of an analog signal corresponds to a transformation of the state-space $\mathbb{X}$ of the signal into Boolean values using a predicate such that $\mu : \mathbb{X} \rightarrow \mathbb{B}$. After using such predicates over analog signals, the analog signal $a : \mathbb{T} \rightarrow \mathbb{X}$ is transformed into a Boolean signal $w : \mathbb{T} \rightarrow \mathbb{B}$.

Early AMS assertion languages incorporate comparison operators to booleanize signals. When signals are close to each other, small variations in signals can lead to many transitions. We observed that these rapid transitions at the outputs, usually called as glitches, affect the quality of verification. For example, an assertion that states *"After signal p exceeds a threshold, p remains stable above the threshold for a period of time"* is susceptible to false negatives because of the glitches around the transition region caused by fluctuations on the signal p.

To prevent glitches at the Boolean output, we propose a *dual-threshold Booleanization* technique. Although it is novel to use dual-thresholds to booleanize real-valued signals in AMS assertion languages, the fundamental idea corresponds to a well-known physical phenomenon, called the hysteresis, and it has been widely employed in electronics design such as analog comparators and Schmitt triggers. In dual-threshold Booleanization, switching thresholds for low-to-high and high-to-low transitions are separate from each other. Therefore, the output retains its value until the input sufficiently changes. It is an effective method to prevent rapid transitions, and it provides an immunity to fluctuations on the signal in booleanization process.

In Figure 2, we show an example scenario for the property stated above. In Column A of the figure, a noisy signal is booleanized by a

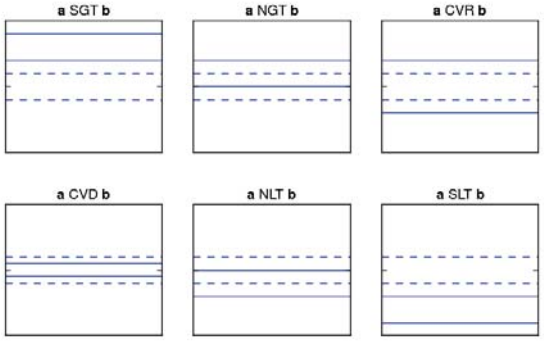

Fig. 1. All six possible relations between halo *a* (solid line) with halo *b* (dashed line). *a* op *b* evaluates true for all points on the time-axis in each subplot.

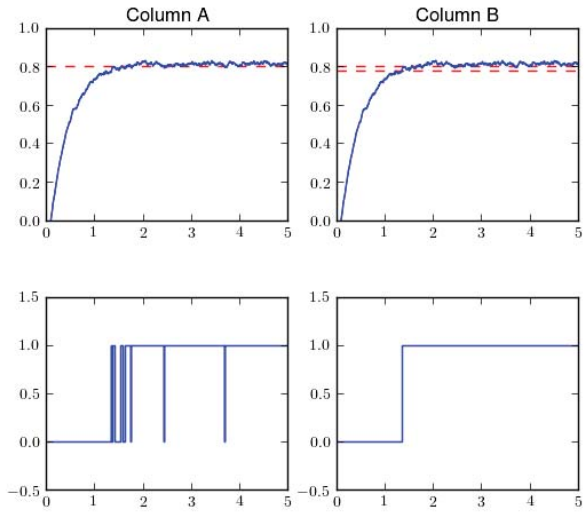

Fig. 2. Column A illustrates a single-threshold booleanization where the output at the bottom is a glitchy boolean. Column B shows booleanization with dual-thresholds and the output is glitch-free.

single threshold. Since there is noise on the signal, a straightforward comparison with a single threshold leads to a Boolean signal with glitches as shown at the bottom of Column A. In Column B of the figure, we show the booleanization of the same noisy signal by comparing it with a dual-threshold. The resulting Boolean signal is now glitch-free, and satisfies the property. The scenario in Column B is closer to the real life evaluation of analog signals.

We think built-in comparison operators using dual-threshold Booleanization will be beneficial in AMS assertion languages. Such an approach provides more natural information transfer from analog domain to Boolean domain.

VI. Analog Measurements

Analog designers are used to writing measurement statements, which can be seen as analog equivalent of an assertion, to evaluate simulation results and automate waveform inspection. Most numerical simulators and waveform analyzers provide support for analog measurements such as crossing, rise-time, fall-time and slew-rate calculations. The integration of these constructs into the analog layer would increase the quality of analog verification and adoption rate of AMS assertion languages by the analog community. By this way, richer analog properties can be defined in assertion languages and waveform measurements become a part of a unified analog verification environment.

We implemented six types of analog measurements, namely $levels$, $amplitude$, $crossing$, $risetime$, $falltime$ and $slewrate$. These are selected because they are measurements for time domain, and fit well in AMS assertion languages. We integrated these measurements into our analog layer and used CosmoScope Reference Manual [14] as the base for definitions of these measurements.

Automatic level calculation for analog signals has an important role for analog measurements. The topline and the baseline levels are required for calculating rise-time and fall-time. Other levels such as mid-line level, 10%-level and 90%-level can be derived from the topline and the baseline levels. Topline and baseline calculation is visualized in Figure 3. To calculate levels, we use the histogram of

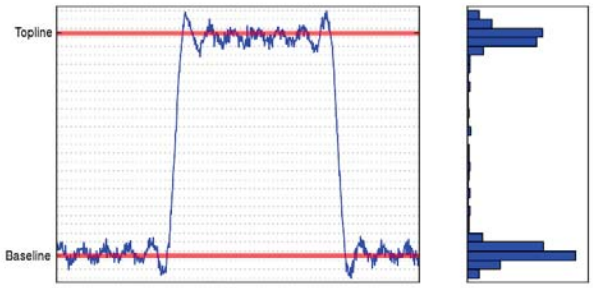

Fig. 3. Automatic level calculation using histogram method

the analog signal, and look for the most occupied bins to determine the topline and the baseline levels. Difference between the topline and the baseline levels corresponds to the amplitude of the analog signal.

Crossing measurement detects intersections of the signal with a constant level or another signal. Crossing measurements can be given a direction such as rising, falling or both directions. It internally uses dual-threshold Booleanization so that we can capture crossing events without false measurements.

Rise-time calculation measures the time passed from the crossing event at the lower level to the crossing event at the upper level in rise direction. Similarly, fall-time calculation measures the time passed from the crossing event at the upper level to the crossing event at the lower level in fall direction. The most common choices for upper and lower levels are 90% and 10%-levels, respectively, although the user can specify custom levels as well.

The slew rate is calculated as the difference between the upper and lower levels of a waveform divided by the rise-time or fall-time of the edge. In Figure 4, we show a slew-rate measurement for an analog signal (shown in the first plot), $G(slewrate(signal) > 34)$, that is, always the slewrate of the analog signal is greater than 34. The second plot shows the slewrate calculation and the third plot shows the boolean output of the comparison, and then the last plot shows the value of the temporal operator G, where the property is false.

VII. Experimental Results

In this section, we present a 10-bit two-stage pipelined analog-to-digital converter (ADC) design as the case study. It shows that how an assertion-based methodology applied for AMS designs in general and that how the extended analog layer can be useful in

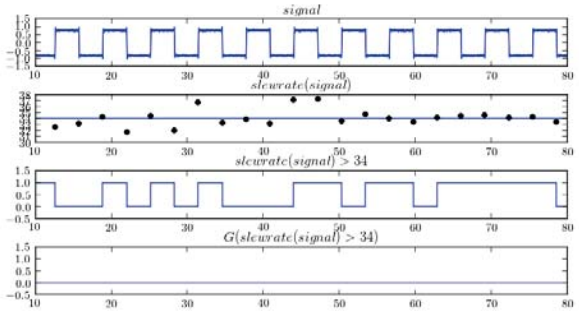

Fig. 4. Slew rate measurement and its integration with assertions

assertion-based AMS verification. Three experiments carried out using the ADC design for this paper. All these experiments address the verification of a property which is repetitive, and cumbersome for manual verification. For example, the set of assertions in the Experiment 2 checks if transition time from one level to another level is acceptable. Because there exist 32 different levels in that case, it means $32^2 = 1024$ transition possibility, and all of them should be verified for full coverage. Verifying all these transitions manually would be very tedious task so designers can only verify a few cases (probably most likely to be failed cases). Therefore, they cease the verification after some point and assume there is no problem in other cases. It constitutes a major weakness for manual verification and this can be improved by assertion-based methodology. Even if you have to simulate all cases separately for full coverage in assertion-based verification, assertions would automatize evaluation of simulation results, allow more cases to be checked compared to manual verification, and eliminate the manual effort in the verification flow. In comparison with formal verification of AMS designs, we do not need to model the ADC design in our flow, and we practically check properties from simulation results without suffering huge state space of large AMS designs.

To summarize, we used a verification flow in our experiments as follows: First, we obtained properties from the designer to be verified in the ADC design. We formalized these properties in our assertion language. We then checked these assertions using our prototype AMS assertion checker tool. Our tool is based on continuous-time notion, and it can handle different signal sampling rates in the same assertion. Linear interpolation is used to calculate the values between sample points in our tool. Property checking by assertions is performed off-line on signals generated from the simulations of various design formats including SPICE or Verilog-AMS.

We simulated the ADC design in Eldo SPICE simulator using transient (.tran) and transient noise (.noisetran) analysis. Noisetran is used to estimate the effect of the noise on the circuit, and this analysis becomes necessary when feature sizes decrease, frequencies increase and supply voltages are lowered.

In the ADC design, the input voltage v_{in} is first sampled and held steady by a sample-and-hold circuit, obtaining the signal *SH*. For the first stage of the pipeline, a flash ADC converts signal *SH* to a 5-bit digital value, which is the first 5-bits of overall ADC output, out_{ADC}. The 5-bit value is then fed to a digital-to-analog converter obtaining signal *DAC*, and this signal is subtracted from v_{in}. This residue is then fed to the second stage of the pipeline to obtain the last 5-bits of out_{ADC}.

In Figure 5, we show analog signals, *SH* and *DAC*, and trigger signals, *SHTRG* and *DACTRG*; which are used in the ADC design experiments. The signals *SHTRG* and *DACTRG* are trigger signals for sample'n hold and digital-to-analog converter circuits, respectively. Time bounds are given as nanoseconds in assertions we define in experiments.

In our experiments, we approximately spend 1s to check the assertion in the first experiment and 1.2s for assertions in second and third experiments. In comparison, AMS simulation takes 2 minutes to compute all output waveforms of the ADC circuit on the same machine. Therefore, assertions we explained in next subsections have an overhead around 1%.

A. Experiment 1: ADC-DAC Operation

The first property that we want to check is the combined operation of 5-bit ADC and DAC circuits, which divide the input voltage range by 32 levels. For a given level number n, we define the *n*th interval $intv_n$ as the range of values between the *n*th level value lvl_n and the following level value lvl_{n+1}. For the correct operation, if the value of *SH* signal is inside $intv_n$ after the trigger *SHTRG*, then at the next cycle the value of *DAC* signal should be equal to lvl_n. Note that we use haloes to express intervals and tolerances around analog signals. We show this property visually in Figure 6. The shaded blue regions show the intervals $intv_{07}$ and $intv_{15}$, the shaded green regions show the levels lvl_{07} and lvl_{15} with haloes. For example, when the value of *SH* is inside $intv_{07}$ after the trigger *SHTRG*, then at the next cycle, the value of *DAC* should be inside the halo of level lvl_{07}.

We divide this property into three sub-properties. First, we formalize the sub-property, spa_n, that *SH* is inside $intv_n$. We use haloes to write this sub-property as follows:

$$spa_n = (\mathcal{H}(SH) \text{ CVD } intv_n) \vee (\mathcal{H}(SH) \text{ NGT } intv_n)$$

We check if *SH* is covered by or is nearly greater than the interval, $intv_n$. NGT relation is also used because small fluctuations crossing interval boundaries should not affect the result. Since the ADC design is sensitive to approximately 1mV changes at the input voltage, we choose the tolerances to calculate haloes accordingly.

The second sub-property, spb_n, checks whether *DAC* signal is currently at the lvl_n. We write this sub-property as follows:

$$spb_n = \mathcal{H}(DAC) \text{ CVR } \mathcal{H}(lvl_n)$$

Finally, we capture the timing specification between two sub-properties in the temporal layer as follows.

$$f_n = G((SHTRG \wedge F_{[20:40]}G_{[0:10]}(spa_n)) \rightarrow F_{[80:120]}G_{[0:50]}(spb_n))$$

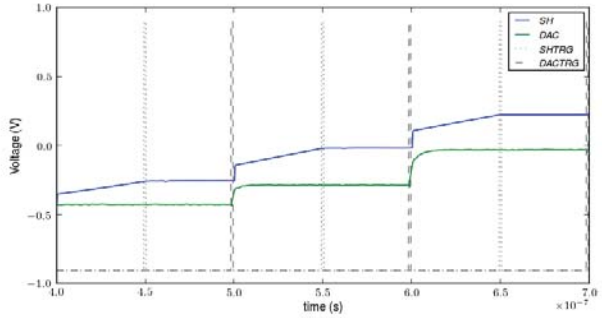

Fig. 5. Signal waveforms from the ADC design used in experiments

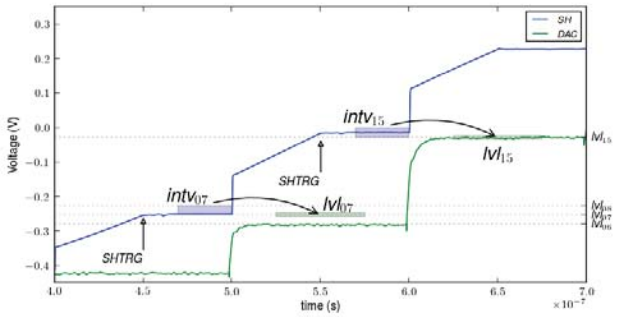

Fig. 6. Visual explanation of the property in Experiment 1

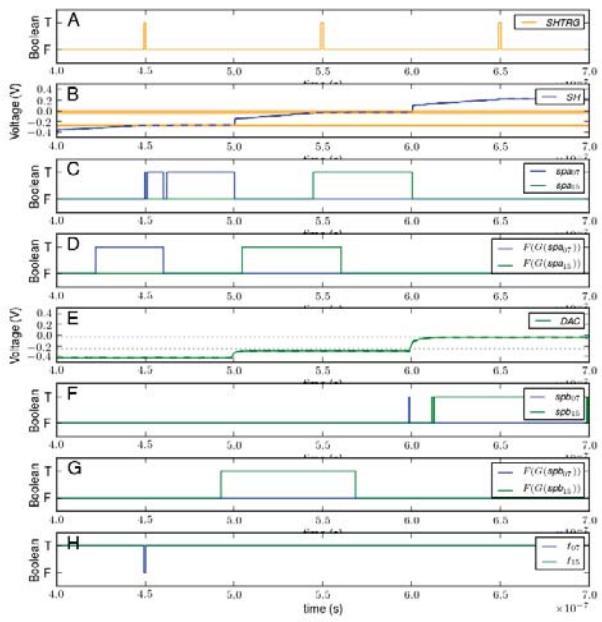

Fig. 7. Evaluation of the property in Experiment 1

This property checks that if *SHTRG* is true and spa_n stays true for 10ns starting between 20ns and 40ns after *SHTRG* is true, then spb_n stays true for 50ns starting between 80ns and 120ns after *SHTRG* is true.

In Figure 7, we show step-by-step evaluation of f_7 and f_{15}. The plot A displays *SHTRG*. In plot B, *SH* and the boundaries of $intv_{07}$ and $intv_{15}$ are shown. Plot C displays Boolean signals spa_{07} and spa_{15}. In plot D, we show Boolean signals $F_{[20:40]}G_{[0:10]}(spa_n)$ and $F_{[80:120]}G_{[0:50]}(spb_n)$. Plots E, F, G are defined similarly for *DAC*. Finally, in plot H, f_{15} is true whereas f_{07} is false. f_{07} is false because *DAC* signal is not equal to lvl_{07}, which can be also seen in Figure 6.

We see that built-in support for haloes allows us to define new relations between signals considering their associated tolerances. Expressing tolerance values with haloes clarifies user-defined allowed regions for analog signals. By this way, automatic evaluation of simulation results can be performed according to designers' intent.

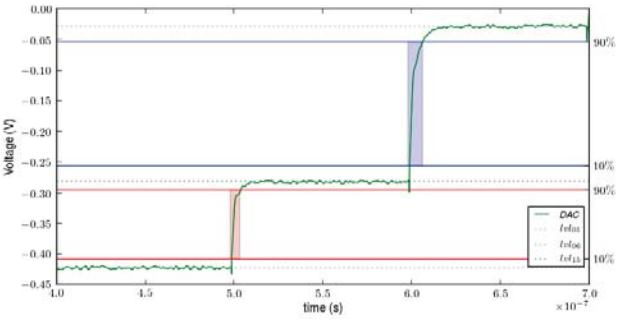

Fig. 8. Visual explanation of the property in Experiment 2

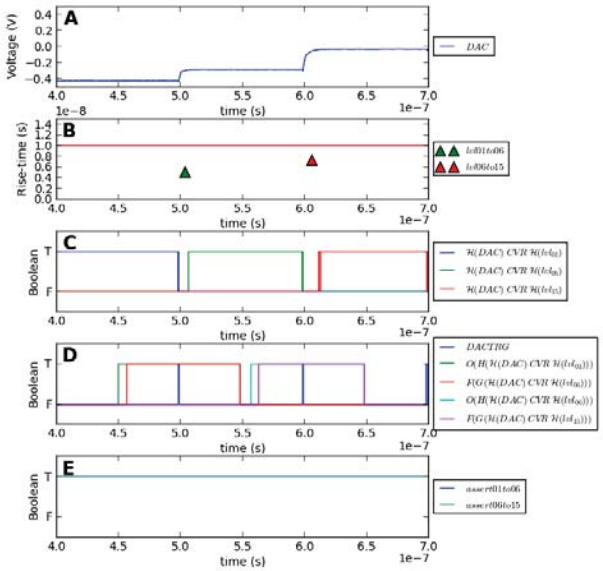

Fig. 9. Evaluation of the property in Experiment 2

Also separating tolerance specification from the property specification helps to keep the size of assertions manageable.

B. Experiment 2: DAC Rise Time

The second experiment shows the integration of analog measurements into assertions. As the second property, we check if the *DAC* signal rises from one level to another level in a specified time, which is shown in Figure 8. The red lines and blue lines denote 10% and 90% levels for the rise-time measurement from lvl_{01} to lvl_{06} and from lvl_{06} to lvl_{15}, respectively. The width of red and blue shaded regions denote the rise-time. The first sub-property $plvl_n$ checks whether *DAC* signal was at lvl_n at least for 50ns ending between 25ns and 50ns before the current time. Similarly, the second sub-property $nlvl_m$ checks whether *DAC* signal will be at lvl_m at least for 50ns starting between 25ns and 50ns after the current time.

$$plvl_n = O_{[25:50]}(H_{[0:50]}(\mathcal{H}(DAC)\ \text{CVR}\ \mathcal{H}(lvl_n)))$$

$$nlvl_m = F_{[25:50]}(G_{[0:50]}(\mathcal{H}(DAC)\ \text{CVR}\ \mathcal{H}(lvl_m)))$$

Then, the third sub-property *meas* captures that the rise-time of *DAC* from lvl_n to lvl_m is less than 10 nanoseconds.

$$meas_{nm} = F_{[0:10]}(risetime(DAC, lvl_n, lvl_m) < 10\text{ns})$$

Finally, these sub-properties are combined as below.

$$f_{n-to-m} = G((DACTRG \wedge plvl_n \wedge nlvl_m) \rightarrow meas_{nm})$$

This property checks that if *DACTRG*, $plvl_n$, and $nlvl_m$ are true, $meas$ is true.

In Figure 9, step-by-step evaluation of f_{01to06} and f_{06to15} is shown. In plot A, we display the *DAC* signal. As seen in the plot A, there are two rise-time cases, the first from lvl_{01} to lvl_{06} and the second from lvl_{06} to lvl_{15}. In plot B, triangles represent rise-time measurements from lvl_{01} to lvl_{06}, and from lvl_{06} to lvl_{15} for the *DAC* signal. The value of these rise-time measurements should be

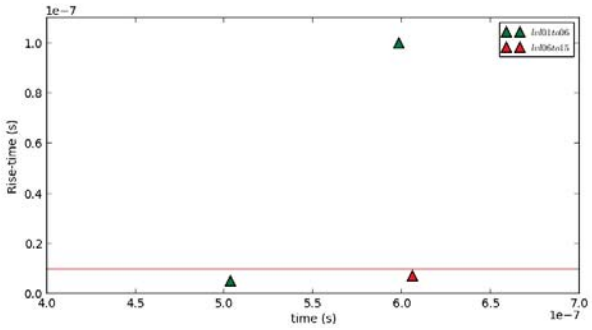

Fig. 10. Rise-time measurement in Experiment 3 without using dual-threshold Booleanization

less than the threshold value (10ns) shown as the red line. In plot C, we check at which level the *DAC* signal is. In plot D, we display Boolean signals $plvl_{01}$, $nlvl_{06}$, $plvl_{06}$, and $nlvl_{15}$. Finally, in plot E, we see that both f_{01to06} and f_{06to15} are true, that is, *DAC* signal satisfies the rise-time condition.

C. Experiment 3: DAC Rise Time without dual-threshold

In Experiment 2, the rise-time operator internally uses dual-threshold booleanization to detect crossings for the 10% and the 90% levels. Experiment 3 checks the same property in Experiment 2 except the rise-time measurement uses a single threshold to detect crossings.

Figure 10 is similar to Figure 9B except we now show the resulting rise-time measurement without dual-threshold booleanization. Note that the scale in Figure 10 is different from the scale in Figure 9B. It can be seen that there is one extra rise-time measurement exceeding the specified rise-time value. We found that this mistake is caused by a spike crossing the 90% level of the rise-time measurement from lvl_{01} to lvl_{06}, that is visible in Figure 8. We see that dual-threshold booleanization can eliminate this type of irregularities, which are common in analog designs, and provide more robust assertion-based verification for AMS designs.

VIII. Conclusion

Assertion-based verification (ABV) has been adapted from the digital domain for analog and mixed signal (AMS) designs in recent years. Due to its digital origins, the built-in support for most low-level aspects of analog designs is not included in early AMS assertion languages. Low-level analog facts can cause false evaluation of simulation results in the assertion-based verification of AMS designs. It means we don't have an effective assertion-based methodology if we don't handle such low-level analog facts. In this paper, we presented three analog layer extensions to improve the low-level analog expressiveness of AMS assertion languages. Specifically, we provide halo, dual-threshold booleanization, and analog measurement extensions. These extensions allow us to express tolerances of analog signals, to convert analog signals into Boolean signals without glitches, and to check common analog numerical routines. We developed a prototype tool to validate our extensions, and show the usefulness of handling analog facts in lower levels without adding significant overhead. Our proposed extensions are a step forward for more expressive analog support in AMS assertions. We think that better support for analog constructs can increase the adoption rate of ABV methodology in the analog design community. In future, we plan to provide support for AC analysis and signal processing algorithms.

Acknowledgment

This work was supported in part by Semiconductor Research Corporation under task 2082.001, Marie Curie European Reintegration Grant within the 7th European Community Framework Programme, and the Turkish Academy of Sciences.

References

[1] E. Barke, D. Grabowski, H. Graeb, L. Hedrich, S. Heinen, R. Popp, S. Steinhorst, and Y. Wang, "Formal approaches to analog circuit verification," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2009, pp. 724–729.

[2] S. Steinhorst and L. Hedrich, "Model checking of analog systems using an analog specification language," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2008, pp. 324–329.

[3] ——, "Equivalence checking of nonlinear analog circuits for hierarchical AMS system verification," in *Proceedings of the Conference on VLSI and System-on-Chip (VLSI-SoC)*, 2012, pp. 135–140.

[4] G. Al-Sammane, M. Zaki, and S. Tahar, "A symbolic methodology for the verification of analog and mixed signal designs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2007, pp. 249–254.

[5] S. Little, D. Walter, K. Jones, C. Myers, and A. Sen, "Analog/mixed-signal circuit verification using models generated from simulation traces," *International Journal of Foundations of Computer Science*, vol. 21, no. 2, pp. 191–210, 2010.

[6] R. Mukhopadhyay, S. K. Panda, P. Dasgupta, and J. Gough, "Instrumenting AMS assertion verification on commercial platforms," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 2, 2009.

[7] S. Lämmermann, J. Ruf, T. Kropf, W. Rosenstiel, A. Viehl, A. Jesser, and L. Hedrich, "Towards assertion-based verification of heterogeneous system designs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2010, pp. 1171–1176.

[8] O. Maler and D. Ničković, "Monitoring temporal properties of continuous signals," in *Proceedings of the Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, 2004, pp. 152–166.

[9] R. Alur, T. Feder, and T. Henzinger, "The benefits of relaxing punctuality," *Journal of the ACM (JACM)*, vol. 43, no. 1, pp. 116–146, 1996.

[10] D. Ulus and A. Sen, "Using haloes in mixed-signal assertion based verification," in *Proceedings of the High Level Design Validation and Test Workshop (HLDVT)*, 2012, pp. 49–55.

[11] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[12] H. Foster, E. Marschner, and Y. Wolfsthal, "IEEE 1850 PSL: The next generation," in *Proceedings of Design and Verification Conference and Exhibition (DVCON)*, 2005.

[13] D. Ničković, "Checking timed and hybrid properties: Theory and applications," Ph.D. dissertation, Joseph Fourier University, 2008.

[14] "Cosmoscope reference manual," Synopsys, 2004.