Run-time Verification using the VHDL-AMS Simulation Environment

Zhi Jie Dong, Mohamed H. Zaki, Ghiath Al Sammane, Sofiène Tahar and Guy Bois*
ECE Dept., Concordia University
*Génie Informatique, École Polytechnique de Montreal
Montreal, Québec, Canada
Email: {zh_do, mzaki, sammane, tahar}@ece.concordia.ca, guy.bois@polymtl.ca

***Abstract*— Analog and Mixed Signal (AMS) designs are an important part of embedded systems that link digital designs to the analog world. Due to challenges associated with its verification process, AMS designs require a considerable portion of the total design cycle time. In this paper, we propose a run-time verification approach for VHDL-AMS designs. The essence of this approach is the construction of timed automata from the given specification. Such automata used as monitor, when interfaced with the VHDL-AMS simulator, detect whether the property of interest is violated or satisfied by a simulation trace. For illustration purposes, we applied the approach using VHDL-AMS simulation environment for the verification of a PLL design.**

I. Introduction

Analog and Mixed Signal (AMS) Hardware Description Languages (AMS-HDLs) (e.g., VHDL-AMS [4]) were developed to provide a unified and coherent modeling and verification environment for AMS designs. Such designs are an important part of embedded systems that link the digital components to the analog world. One advantage offered by such AMS-HDLs is the fact they allow the design and verification at different levels of abstraction supporting a structured design methodology. They help choose the appropriate modeling approach, whether a structural description, a behavioral description, or a combination of the two. The modeling is not restricted to AMS systems but also other systems such as mechanical and optical.

Classically, the verification of AMS-HDL designs is carried out using simulation. However, such method is not enough to validate interesting properties of the design behavior involving for instance temporal requirements. Such commodity is usually offered by formal verification techniques, like model checking [5], which aim to prove that a circuit behaves correctly by satisfying a set of temporal properties. Unfortunately, model checking AMS designs is computationally expensive and therefore suffers even more from the state-space explosion problem that makes exhaustive verification very hard [5].

With the goal to achieve confidence requirements in the design process and to tackle the drawbacks of the above-mentioned verification techniques, the integration of both simulation and formal methods was advocated. One such integration is known as *run-time verification* [2]. Run-time verification is a technique for checking whether an execution of the design model under verification violates the given properties. The AMS design is simulated by attaching it to a testbench which provides the inputs while monitoring its output by means of logical models that describe the property of interest, in order to detect possible errors. The monitor could be as simple as observing the current or voltage at different nodes of the design. The main challenge in monitoring AMS designs is the development of adequate monitors able to express the properties.

In this paper, we propose a run-time verification approach for VHDL-AMS designs. The essence of this approach is the construction of timed automata from the given specification; Such automata, when interfaced with the VHDL-AMS simulator, monitor whether the property of interest is violated or satisfied by a simulation trace.

The rest of the paper is organized as follows: We start by providing a discussion about related work in Section II. In Section III, we describe the proposed verification methodology. A PLL design is used as illustrative example in Section IV, before concluding with a discussion in Section V.

II. Related Work

According to [8], run-time verification can be classified in two different groups: offline and online monitoring. In *offline monitoring*, the property verification starts after the whole simulation trace is provided, while in *online monitoring*, the monitoring is interleaved with the simulation process. The advantage of online monitors is the ability to detect violation or satisfaction as soon as they happen. Several works have been recently proposed for run-time verification of AMS designs.

In [9], an online monitoring technique was proposed, where the authors used linear hybrid automata as template monitors for the time domain features of oscillatory behavior (e.g., jitter). The monitors are implemented within the *PHAver* tool, where the design nonlinear equations are approximated with piecewise differential models. A similar work was proposed in [10], where the authors used interval arithmetics based methods to generate the solution flow for the design equations. However, instead of creating template automata to use as monitor, they build a timed automata monitor from timed computational temporal logic (T-CTL) properties. Such monitor can then detect bad behaviors within a specified time period

1-4244-1164-5/07/$25.00 ©2007 IEEE.

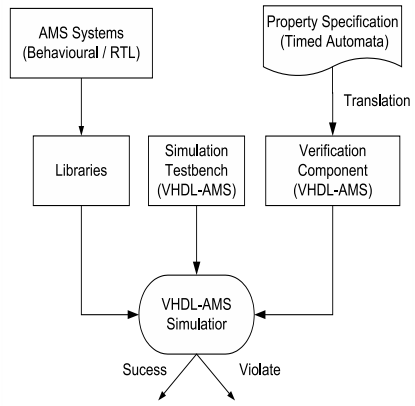


Fig. 1. Verification Methodology

of the interval arithmetics simulation. The work in [9] and [10] consider only analog behaviors, while mixed systems are not supported. In addition, they are computationally expensive, due to the requirement to build a solution flow for the system equations, making them only adequate for small designs.

In [7], Maler *et.al* proposed an offline methodology for monitoring the simulation of continuous time signals described by differential equations. The work is based on extending the PSL (Property Specification Language) [1] logic to support monitoring analog signals. A synthesized timed automata is then used to monitor Matlab simulation traces to check for property violation. The main drawback of the approach is that violation cannot be checked unless the whole simulation is finished, which consumes time and memory resources.

Our methodology is close to the one proposed by Tan *et. al* in [6], where the authors developed tools for online monitoring of hybrid systems using timed and linear hybrid automata to monitor hybrid system behavior. However, we distinguish ourselves in the following. We build the monitors as timed automata in the VHDL-AMS environment along with the AMS design to verify. Therefore, we do not need to do any translation which can be quite expensive like the transformation of timed automata monitors into hybrid automata, etc.

III. Methodology

The proposed run-time verification for AMS designs is illustrated in Figure 1. Given an AMS design description written in VHDL-AMS, we describe the properties to be verified as timed automata, which are translated to VHDL-AMS as connected components. The monitoring components together with the VHDL-AMS design libraries and the simulation test bench are then input to the ADM VHDL-AMS simulator [3] to perform run-time verification.

In the remaining of this section, we will briefly overview the VHDL-AMS simulation and give the description of the monitors and how they can be used to detect properties of the VHDL-AMS design.

A. VHDL-AMS

VHDL-AMS is an extension of the IEEE standard VHDL language, which allows the unified modeling of the analog and digital parts of the design. For the simulation of such models, a synchronization between the discrete-time and the continuous-time simulators engines to handle mixed-signal interactions (e.g., threshold crossings) is usually required. In fact, the simulation of discrete-time models relies on event-driven techniques for which the states of the model are reevaluated when signals change their values. On the other hand, the simulation engine of continuous-time models uses numerical techniques to solve the set of differential equations describing the analog behavior. The differential equations are discretized using a fixed or a variable time step $\tau = t_{i+1} - t_i$.

We can interpret a simulation trace of the VHDL-AMS model as a sequence of tuples: $\Sigma = \langle s_0, \nu_0, \mathcal{T}_0 \rangle \ldots \langle s_f, \nu_f, \mathcal{T}_f \rangle$, where s_i is the valuation of the discrete signals during time interval $\mathcal{T}_i = [\mathcal{T}_i^l, \mathcal{T}_i^r]$, with $\mathcal{T}_i^r = \mathcal{T}_{i+1}^l$, ν_i is the valuation of the continuous variables at $\mathcal{T}_i$, where 0 and f are the starting and ending time intervals $\mathcal{T}$ specified for the whole simulation trace and $0 \leq i \leq f$.

B. Monitoring Timed Automata

A *Monitoring Timed Automaton* (MTA) [6] is a deterministic finite states automaton, with a set of timers constraints for each state location. A timed automaton is said to be deterministic if only one transition can be taken at a time from any location s_i. Formally, an MTA $\mathcal{M}$ is a tuple $\{S, S_0, V, Q, \Theta, G, T, SC, \mathcal{C}\}$ where S is a finite set of locations, $S_0 \subseteq S$ is the finite set of initial locations, V is a set of state variables, $Q : S \rightarrow 2^{\Sigma}$ is a labelling function assigning each location a set of atomic propositions $\{p | p \in \Sigma\}$ over state variables v_j, $v_j \in V$. Θ is the set of clocks with time valuation (timer assignment) $\delta : \Theta \rightarrow \mathbb{R}$. $G : S \rightarrow 2^{TC(\Theta)}$ assigns to each location a set of timers constraints (TC) over the location timers Θ. For example, the satisfaction of a timer constraint $x > 0$ (where $x \in \Theta$) by a timer valuation δ is defined as follows: $\delta \models x > 0$ iff $\delta(x) > 0$. $T \subseteq S \times S$ is the set of edges between locations and $SC : T \rightarrow 2^{\Theta}$ associates with each edge a set of clocks that need to be set. The monitoring automata is also extended with the acceptance condition that determines the set of allowable states; therefore rejecting bad states, i.e., the automata is extended with the acceptance set $\mathcal{C}$, such that $s \in \mathcal{C} \subseteq S$ if there is no path from s to any $s' \in \mathcal{B}$, where $\mathcal{B} \subseteq S$ is the *Büchi* acceptance condition.

Consider the digitally controlled sine wave generator (designed using a typical Wien-Bridge oscillator) as shown in Figure 2(a). Such circuit can generate sine wave signals at three different frequencies in a sequence determined by the digital controller. For instance, by setting the NMOS switches M_{11} and M_{12} *on* and the other switches off, the output will be a sine wave signal with frequency f_1. Suppose the specification requires that the analog circuit generates sine wave signals with different frequencies f_1, f_2, f_3, in a repetitive increasing order (e.g., $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_1 \ldots$), where each signal lasts for a maximum period of 10 ms. An interesting temporal property

to check on this design would be: *If the current frequency of the sine wave is f_1, then in the future within a maximum period of* 30 *ms, the frequency will eventually go back to f_1.*

The automaton that models this property is shown in Figure 2(b), where state S_1 refers to output signal where frequency f_1, and the state S_2 refers to output signal with the frequency being either f_2 or f_3. S3 and S4 are the failure states, representing the property violations. T_p represents the signal time period with $T_p = \frac{1}{f}$, and Tp_n is the constant value with $Tp_n = \frac{1}{f_n}$, $1 \leq n \leq 3$.

C. Run-Time Verification

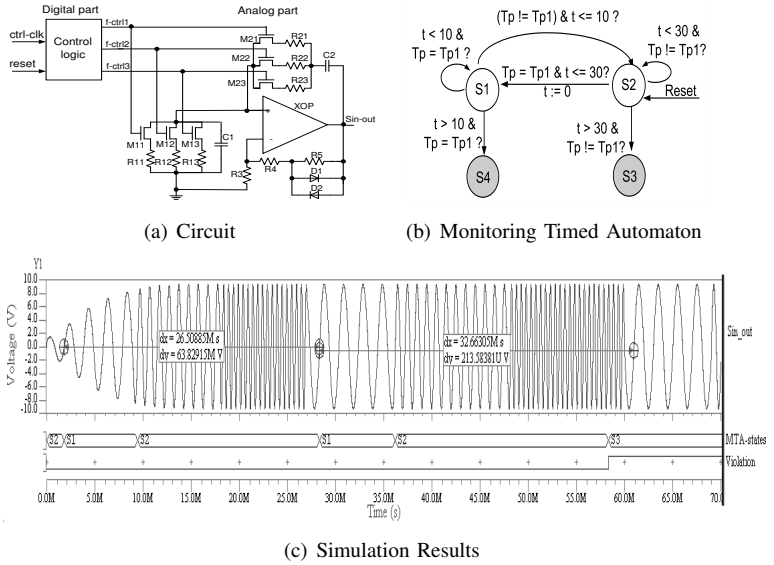
To understand how an automaton accepts a sequence of states, we define the notion of trace. A *trace* of an automaton is a sequence of states together with an associated time counter evaluating the time spent (time interval) at each state location. A *trace* of monitoring timed automata $\mathcal{M}$ is a sequence $\pi = < s_0, \delta_0, \nu_0, I_0 > < s_1, \delta_1, \nu_1, I_1 > \ldots$, where δ_i are evaluations of the clocks upon entering the location s_i, I_i is the time interval representing the time spent at location S_i. The timer associated with each state is a decreasing timer such that we have for each pair $< s_i, \delta_i, I_i > < s_{i+1}, \delta_{i+1}, I_{i+1} >$, $e = (s_i, s_{i+i})$, $\delta_{i+1} = \delta_i - |I_i|$ or $\delta_{i+1}(x) = t \in \mathbb{R}$ for $x \in SC(e)$ [6].

For simplicity, we consider here, only that timed automata accept timed words on the valuation of variables. Hence, we define a *timed word* as a timed state sequence $\eta = \{\nu_0, \tau_0\}, \{\nu_1, \tau_1\}, \ldots$, where $\tau_i = [\tau_i^l, \tau_i^r]$, with $\tau_i^r = \tau_{i+1}^l$ and $\nu_i(t)$ denotes the evaluation of the state variables at time $t \in \tau_i$ [6]. Such timed word could be interpreted as the VHDL-AMS simulation sequence of states Σ mentioned earlier. We say that MTA $\mathcal{M}$ accepts a finite timed word η if $\mathcal{M}$ has an accepting trace π for η. More formally, a timed word $\eta = \{\nu_0, \tau_0\}, \{\nu_1, \tau_1\}, \ldots$ is accepted by an automaton $\mathcal{M}$, if there is a trace of $\mathcal{M}$: $\pi =< s_0, \delta_0, I_0 >< s_1, \delta_1, I_1 > \ldots$, such that if $t \in \tau_i$, then there is an l such that $t \in I_l$ and $\nu_i(t) \models p \Rightarrow p \in Q(s_l(t))$, where $Q(s_l(t))$ is the labelling function for location s_l.

The verification results are shown in Figure 2(c). Inspecting the *Violation* simulation trace, we notice that there is no violation detected during the first cycle where each signal lasts 9 *ms*. The $MTA-states$ trace shows the transition between the automaton states. In the first cycle, The automaton changes states from S_1 to S_2 and back to S_1 within 30 *ms*. However, during the second cycle the monitor reports a violation as shown by the *Violation* trace. In that case, we see that the $MTA-states$ indicate that the automaton reaches a failure state, i.e., S_3. The reason of such violation is that the digital controller allows the frequency of the output signal to change back to f_1, only after 32 *ms*.

IV. PLL BASED FREQUENCY SYNTHESIZER DESIGN

We applied our verification methodology to a PLL design based frequency synthesizer. Figure 3(a) shows a block diagram for the frequency synthesizer which consists of five blocks: comparator, voltage-controlled oscillator (VCO), digital phase and frequency detector (PFD), analog loop filter



(a) Circuit (b) Monitoring Timed Automaton

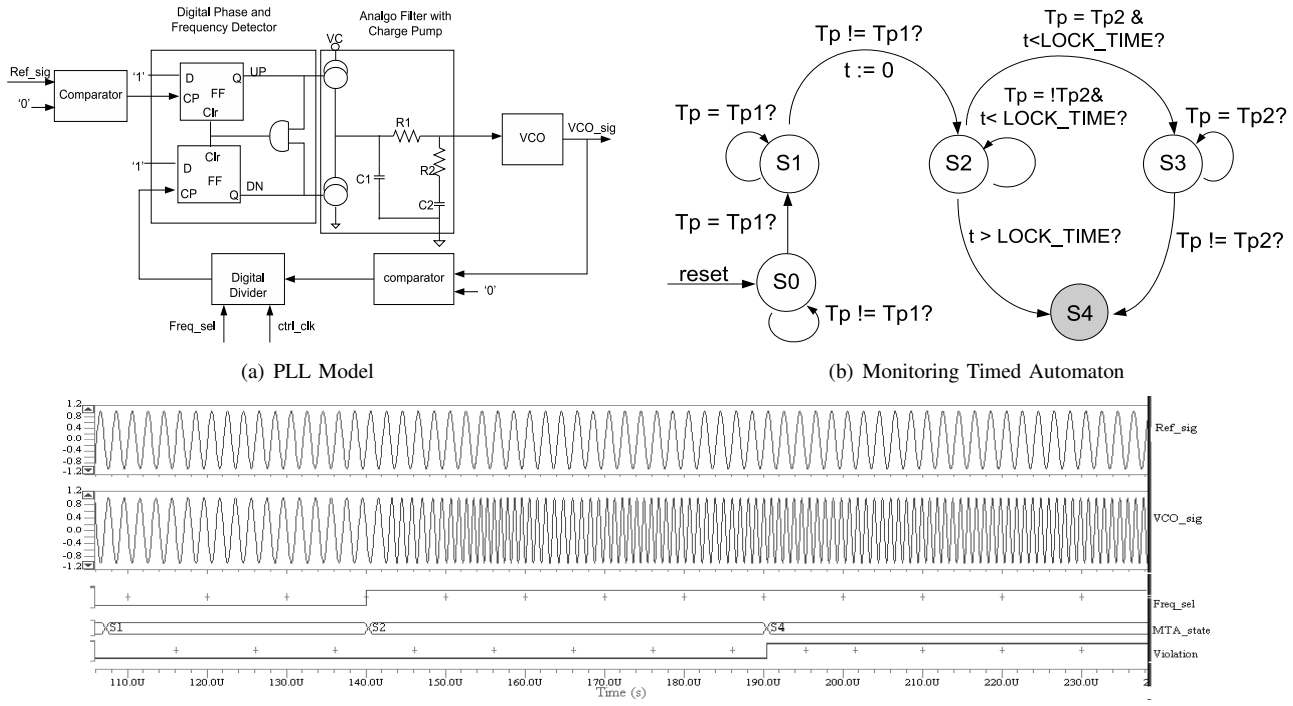
(c) Simulation Results

Fig. 2. Illustrative Non-linear Analog Circuit

with charge pump and digital divider. The PFD is composed of two D Flip-Flops followed by a charge pump and an analog passive lead-lag loop filter. The input reference clock is a sine wave signal with frequency W_0. The VCO output is a cosine wave signal with frequency N times of W_0. If the frequency control signal *Freq_sel* is set to 0, N will be equal to 1 and the frequency of the reference clock input and VCO output will be the same. If *Freq_sel* is set to 1, N will be equal to 2 and the VCO output frequency is two times of the reference input clock.

The *lock time* is one of the main issues in the verification of PLL based frequency synthesizer. It is the time necessary for the synthesizer to switch from one frequency to another within a given range. Such property is important because during the lock time, no data can be transmitted, so having a larger lock time can reduce the data rate of the system. The property to be verified is described as follows; *if the Freq_sel changes, the VCO output signal should be changed to a new frequency within the lock time*. The corresponding automaton is shown in Figure 3(b), where S_0 is the initial state after reset. S_1 is the state referring to VCO output signal with frequency f_1. S_2 is the transition state referring to VCO output signal with frequency changing from f_1 to f_2. S_3 is the state where the VCO output signal frequency is equal to f_2. S_4 is the failure state representing property violation. If the VCO output signal frequency has not changed to f_2 after the lock time (*LOCK_TIME*), the monitor will reach state S_4 and the violation of the property will be reported. T_p represents the signal time period with $T_p = \frac{1}{f}$ and T_{p_n} is the constant value with $T_{p_n} = \frac{1}{f_n}$.

The simulation result is shown in Figure 3(c). The reference clock input signal *Ref_sig* is a sine wave signal with a constant frequency of 500 KHz. VCO's central frequency is 500 KHz with the VCO control signal; *Filter_out* equals to 0. Assuming



(a) PLL Model

(b) Monitoring Timed Automaton

(c) Simulation Results

Fig. 3. Frequency Synthesizer

the design specification requires the *LOCK_TIME* to be 50 μs, we will verify whether the VCO output signal frequency can change from 500 KHz to 1 MHz within 50 μs. If it is true, the property is satisfied. Otherwise, it is violated and a violation will occur at time $T_1 = T_0 + LOCK\_TIME = 190$ μs. From the simulation we can see that the actual *LOCK_TIME* is around 100 μs, where the voltage of *Filter_out* is equal to 2.0V. Inspecting the *Violation* simulation trace, we notice that there is a violation detected at time 190 μs, where the frequency of VCO output signal is not equal to 1 MHz and MTA reaches to the failure state S_4.

V. Conclusion

In this paper, we have presented a practical run-time verification methodology for AMS designs. The approach is based on modeling and simulating the AMS design along with the monitoring automata in VHDL-AMS environment in an online fashion, hence avoiding state space explosion problems. We have used the methodology to verify the PLL locking time of a frequency synthesizer design.

This process is much more reliable than manual (visual or textual) inspection of simulation traces which will also cost lots of time. On the other hand, combining simulation and temporal properties based verification is an attempt to achieve the benefits of both approaches, while avoiding some of the pitfalls of adhoc simulation and the complexity of formal methods. Future work includes the verification of more complex case studies. In addition, we plan to investigate methods for guiding the input signal test generation using the monitors, which can help in violations detection.

References

[1] Accellera Property Specification Language Reference Manual (2004), http://www.accellera.org

[2] K. Kundert and H. Chang. Verification of Complex Analog Integrated Circuits. In Proc. IEEE Custom Integrated Circuits Conference, pp.177-184, 2006.

[3] ADVance MSTM Reference Manual, Mentor Graphics, http://www.mentor.com

[4] E. Christen and K. Bakalar. VHDL-AMS: A Hardware Description Language for Analog and Mixed-signal Applications. In IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing., 46: 1263-1272, 1999.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking. MIT Press, 2000.

[6] L. Tan, J. Kim, I. Lee: Testing and Monitoring Model-based Generated Program. Electronic Notes in Theoretical Computer Science. 89(2): (2003)

[7] O. Maler, D. Nickovic. Monitoring Temporal Properties of Continuous Signals. In Formal Modelling and Analysis of Timed Systems, LNCS 3253, Springer, 2004, pp.152-166

[8] O. Maler, D. Nickovic, Amir Pnueli. Real Time Temporal Logic: Past, Present, Future. In Formal Modelling and Analysis of Timed Systems, LNCS 3829, Springer, 2005, pp.2-16

[9] Goran Frehse, Bruce H. Krogh, Rob A. Rutenbar, Oded Maler: Time Domain Verification of Oscillator Circuit Properties. Electronic Notes in Theoretical Computer Science. 153(3): 9-22, 2006.

[10] M. Zaki, S. Tahar, and G. Bois: A Practical Approach for Monitoring Analog Circuits; Proc. ACM Great Lakes Symposium on VLSI, April 2006, pp. 330-335.