

Recurrence in Dense-time AMS Assertions

Sayandeep Sanyal, *Student Member, IEEE*, Antonio Anastasio Bruto da Costa, *Student Member, IEEE*,
 Pallab Dasgupta, *Senior Member, IEEE*
 Formal Methods Laboratory, Department of Computer Science and Engineering,
 Indian Institute of Technology Kharagpur

Abstract—The notion of recurrence over continuous or dense time, as required for expressing *Analog and Mixed-Signal* (AMS) behaviours, is fundamentally different from what is offered by the recurrence operators of SystemVerilog Assertions (SVA). This article introduces the formal semantics of recurrence over dense time and provides a methodology for the runtime verification of such properties using interval arithmetic. Our property language extends SVA with dense real-time intervals and predicates containing real-valued signals. We provide a tool kit which interfaces with off-the-shelf EDA tools through standard VPI.

Index Terms—Sequence Expressions, Assertions, Recurrence, Analog Mixed-Signal

I. INTRODUCTION

Assertion (property) language standards, such as SVA [1] and PSL [2], are widely used in digital design verification, and some of their recent features enable the specification of properties using real variables, including analog signals sampled at discrete clock boundaries. Properties in SVA are based on *sequence expressions* that capture sequences of Boolean events separated by clock-cycle delays.

The clocked semantics of SVA can lead to precision related problems when dealing with analog signals. For example, consider the property: "*The output V_{out} must cross 1.8 V within $2\mu s$ to $4.25\mu s$ of the input V_{in} crossing 3V*". If the clock, clk , has a period of $0.4\mu s$, we may write this assertion in SVA as follows:

```

wire x, y;
assign x = V(Vin) > 3;
assign y = V(Vout) > 1.8;
property DelayCheck;
  @(posedge clk) $rose(x) |-> ##[5:11] $rose(y);
endproperty
assert property (DelayCheck);
  
```

Note that the real time interval, $[2\mu s : 4.25\mu s]$, is approximated by the discrete interval $[5 : 11]$ in terms of the number of clock cycles of the clock, clk , of period $0.4\mu s$. The loss of precision due to this approximation may lead to missing a failure as shown in Fig. 1. Here V_{in} crossed 3V exactly $4.3\mu s$ after V_{out} crossed 1.8 V, which exceeded the real time interval, but the assertion checker detects that the crossing took place within the specified number of clock cycles.

In general over-approximation (equivalently, under-approximation) produces false positives (equivalently, false negatives). Increasing the precision of the assertion clock reduces the chance of a false positive/negative, but it increases

the number of cycles in the interval, and thereby the assertion checking overhead. For this reason, existing literature, including our own [7–14], advocates the use of dense time assertion checking, which works using real interval arithmetic as opposed to cycle based reasoning.

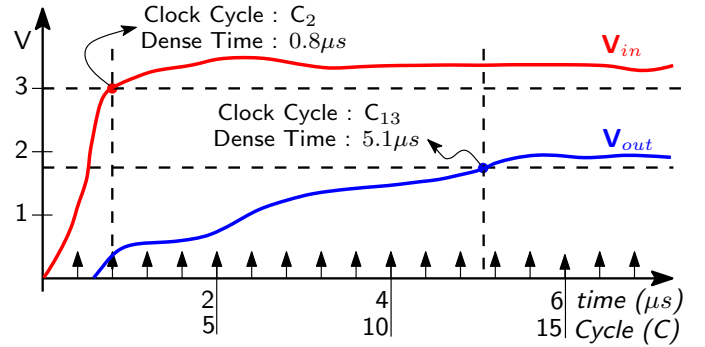


Fig. 1: Bugs can escape due to low precision sampling

The semantics of dense-time naturally allows properties to hold continuously over a dense time period. We define *recurrence* to mean that the truth of an expression holds true continuously over a period of time. For example, consider the requirement, "*If the enable becomes true, and thereafter within 5ms the output voltage is above 3V for at least 2ms, then within the following 0.7ms the out_good signal stays true for at least 1ms.*" An intrinsic feature of such requirements is that predicates must be true continuously over a time period. This is fundamentally different from the notion of recurrence in languages like SVA [1] and PSL [2], where recurrence means a countable non-overlapping series of matches of a sequence expression.

In this article, we propose the dense time semantics of recurrence and our methodology for evaluating assertions having recurrence operators as well as other operators. Since recurrence operators are frequently used with other types of operators in an assertion, we provide the integrated set of interval arithmetic steps used in our tool, CHAMS. Results on CHAMS are also provided at the end.

II. RECURRENCE IN AMS ASSERTIONS

This section demonstrates the use of recurrence operators over dense-time. Consider the waveforms for the voltages of analog nets, a , b , and c , shown in Fig. 2. Also shown are truth intervals of some of the *predicates over real variables* (PORVs), and the match/fail intervals of the assertions described below.

Example 1. If $V(a)$ remains above 1.6 V for 2.3 ms, then $V(b)$ will be above 1.1 V.

$$\Phi_1 : \{V(a) > 1.6\} [*0.0023] | \rightarrow \{V(b) > 1.1\};$$

The authors thank Intel corporation CAD SRS funding for partial support of this research. The authors also acknowledge Antara Ain for building the initial framework for the tool.

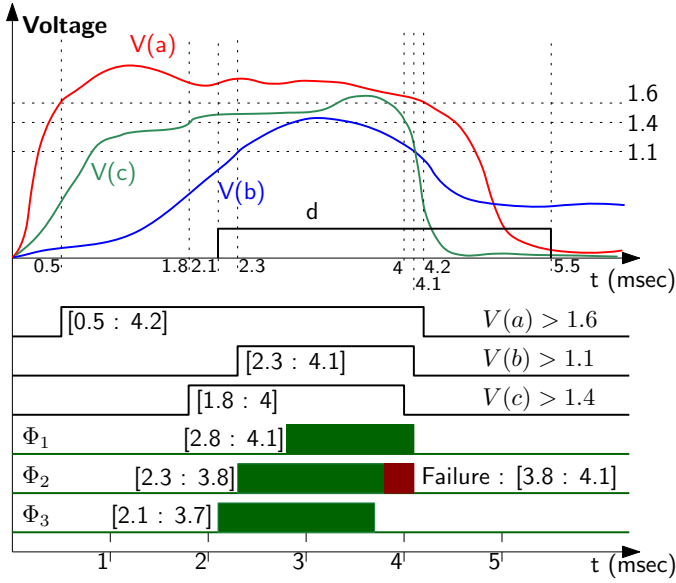


Fig. 2: Waveforms of signals and assertion matches

The syntax of the recurrence operator is similar to that of SVA, but the semantics of recurrence of the PORV, $V(a) > 1.6$, is continuous over the specified dense time. The unit of voltage is volts and the unit of time is seconds.

Example 2. Whenever $V(b)$ is greater than 1.1 V, d will be high at some later time between 0.2 msec and 1.3 msec, and $V(c)$ will remain above 1.4V until d becomes true.

$$\Phi_2 : \{V(b) > 1.1\} \mid \rightarrow \{V(c) > 1.4\} [*0.0002 : 0.0013] \{d = 1\};$$

Example 3. If $V(a)$ is higher than 1.6 V and within 1.3 msec to 2.9 msec $V(c)$ is higher than 1.4 V for 0.3 msec, then $V(b)$ will be greater than 1.1 at some time later between 0.4 ms and 1.2 ms and d will be high until $V(b) > 1.1$.

$$\Phi_3 : \{V(a) > 1.6\} \#\#[0.0013 : 0.0029] \{V(c) > 1.4\} [*0.0003] \mid \rightarrow \{d = 1\} [*0.0004 : 0.0012] \{V(b) > 1.1\};$$

The non-vacuous matches of these assertions are shown in Fig. 2. The assertion φ_2 also has failures. It may be noted that AMS assertions can match or fail continuously over a period of time.

III. FORMAL SEMANTICS

As in SVA, our language, Analog Mixed-Signal Assertion Language (AMSAL), uses the notion of *sequence expressions*. A property in AMSAL takes one of the following forms:

SEQ $\mid \rightarrow$ SEQ
 SEQ $\mid \rightarrow \#\#[a : b]$ SEQ
 where:
 SEQ \rightarrow BEXPR
 $\mid \sim \{\text{BEXPR}\}$
 $\mid \{\text{SEQ}\} [*a]$
 $\mid \text{SEQ} \#\#[a : b] \text{SEQ}$
 $\mid \{\text{SEQ}\} [*a : b] \text{SEQ}$

In the syntax, BEXPR is a Boolean expression over events and PORVs, and $a, b \in \mathbb{R}^+$, $b \geq a$. We use symbol ϕ for Boolean expressions, φ for sequence expressions, and Φ for properties.

Definition 1. Predicate over Real Variables (PORVs) If $X = \{x_1, x_2, \dots, x_n\}$ denotes the set of continuous variables,

then a Predicate over Real Variables, P , may be defined as, $P ::= f(x_1, x_2, \dots, x_n) \sim 0$, where f is a mapping, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and \sim is a relational operator such that $\sim \in \{>, \geq\}$. \square

Other relational operators may be derived using \sim along with appropriate propositional connectives.

Definition 2. Events: Events are of the form $@*(P)$, where P is a PORV and $*$ $\in \{+, -, \}$. $@^+(P)$, $@^-(P)$, $@(P)$ are true respectively at the positive edge, negative edge, both positive and negative edge of the truth of P . \square

Boolean expressions in AMSAL are written over Boolean propositions, PORVs, and events. The syntax for BEXPR is as follows:

BEXPR \rightarrow EVENT
 \mid EVENT, ASGMT
 \mid (CONJUNCT)
 \mid (CONJUNCT), ASGMT
 \mid EVENT && (CONJUNCT)
 \mid BEXPR OR BEXPR
 CONJUNCT \rightarrow CONJUNCT && PORV
 \mid PORV

Definition 3. Satisfaction Relation: $\tau(t) \models \varphi$ is defined as follows for Boolean expressions:

- $\tau(t) \models P$ iff P is a PORV, and P is true for signal valuations in state $\tau(t)$ or P is a Boolean signal and $\tau_P(t) = \text{true}$.
- $\tau(t) \models D$ where $D = P_1 \vee P_2 \vee \dots \vee P_N$ iff $\exists_{1 \leq i \leq N} \tau(t) \models P_i$, and P_i is a PORV.
- $\tau(t) \models C$ where $C = P_1 \wedge P_2 \wedge \dots \wedge P_N$ iff $\forall_{1 \leq i \leq N} \tau(t) \models P_i$, and P_i is a PORV.
- $\tau(t) \models E$ where E is an event on PORV P , iff either of the following holds:
 - $E \equiv @^+(P) : \tau(t) \models P \wedge \exists_{t' < t} \forall_{t \in [t', t)} \tau(t') \not\models P$
 - $E \equiv @^-(P) : \tau(t) \not\models P \wedge \exists_{t' < t} \forall_{t \in [t', t)} \tau(t') \models P$
 - $E \equiv @(P) : \tau(t) \models @^+(P) \vee \tau(t) \models @^-(P)$

\square

We now describe the simulation semantics of recurrence in dense-time as used in AMSAL with an interpretation over a simulation trace τ .

Definition 4. Simulation Trace A simulation trace τ is a mapping $\tau : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^{|V|}$, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of variables (Boolean and Real) representing signals of the system. The state of the system in τ at time t is given as $\tau(t)$. For $x \in V$, its value at time t , in τ , is $\tau_x(t)$. \square

In Definition 5, state satisfaction for a Boolean expression ϕ , that is $\tau(t) \models \phi$, is extended to sequence expressions with support for recurrence with dense-time. Note that, a Boolean expression is also a sequence expression.

Definition 5. Extended Satisfaction Relations: $\tau(t) \models_e \varphi$ is recursively defined for a sequence expression φ , to be true iff:

- $\{\varphi_1\} [*a]$, $a > 0 : (\forall_{t' \in [0 : a]} \tau(t - t') \models_e \varphi_1)$
- $\varphi_1 \#\#[a : b] \varphi_2 : \tau(t) \models_e \varphi_2 \wedge t \in [t' + a : t' + b] \wedge \tau(t') \models_e \varphi_1$
- $\{\varphi_1\} [*a : b] \varphi_2 : \tau(t) \models_e \varphi_2 \wedge \forall_{t' \in [0 : a]} \tau(t - t') \models_e \varphi_1$

where φ_1 and φ_2 are sequence expressions. \square

The satisfaction of a temporal expression has a begin time and an end time for a match of the expression, known respectively as the *begin match* and *end match* for a sequence

expression. The relation \models_e describes the end-match for a sequence expression. Due to the nature of the truth of a temporal property over analog artifacts, an end-match at time t for sequence expression φ may be associated with multiple begin-match time points, and vice versa.

Definition 6. Begin Match: We define $\mathbb{B}(\tau, \varphi, t)$ as the set of time points \hat{t} such that there exists a match of φ in τ starting at \hat{t} and ending at t . Formally, for $\tau(t) \models_e \varphi$, $\hat{t} \in \mathbb{B}(\tau, \varphi, t)$ is said to be a begin match for φ 's end-match at time t iff:

- $\{\varphi_1\}[*a]: a > 0, t' = t - a, \hat{t} \in \mathbb{B}(\tau, \varphi_1, t')$
- $\varphi_1 \# [a:b] \varphi_2$ or $\{\varphi_1\}[*a:b] \varphi_2 : (t' \in \mathbb{B}(\tau, \varphi_2, t)) \wedge (t'' \in [t' - b : t' - a]) \wedge (\tau(t'') \models_e \varphi_1) \wedge (\hat{t} \in \mathbb{B}(\tau, \varphi_1, t''))$

where φ_1 and φ_2 are sequence expressions. \square

Definition 7. Match of an assertion: We say that an assertion has matched at time t iff t is an end match of the antecedent and is a begin match of the consequent. Hence, the assertion $\varphi_1 \mapsto \varphi_2$ matches trace τ non-vacuously at time t , iff $\exists t' \geq t (\tau(t') \models_e \varphi_1 \wedge t' \in \mathbb{B}(\tau, \varphi_2, t))$.

Note that an assertion vacuously match at time t , when $\tau(t) \not\models_e \varphi_1$. An assertion fails at time t iff it has no vacuous nor non-vacuous match at time t . \square

In the following section, we propose an interval abstraction that enables computing the match of assertions over dense time.

IV. INTERVAL ARITHMETIC FOR AMSAL

In this section we discuss how the match of a property interpreted over dense-time may be computed, by recursively interpreting the truth of expressions in the property as operations over time intervals.

Definition 8. Time Interval: A time interval I is a nonempty convex subset of $\mathbb{R}_{\geq 0}$ expressed as $[a : b]$, $(a : c)$, $[a : c)$, and $(a : c]$ where $a, b, c \in \mathbb{R}_{\geq 0}$ and $b \geq a, c > a$. $l(I)$ and $r(I)$ are used to denote the left ends and right ends respectively of interval I . \square

For an interval I , the Minkowski operators are as follows. The Minkowski sum, $I \oplus [c : d]$, where $c, d \in \mathbb{R}_{\geq 0}, c \leq d$, is computed as, $I \oplus [c : d] = [l(I) + c : r(I) + d]$. Similarly, the Minkowski difference, $I \ominus [c : d]$, where $c, d \in \mathbb{R}_{\geq 0}, c \leq d$, is computed as, $I \ominus [c : d] = [l(I) - d : r(I) - c]$. Note that any interval $[a : b]$, where $a > b$ is a null interval.

Definition 9. Truth Interval: Time interval I is a truth interval of φ , where φ is a PORV, event, Boolean signal, or Boolean expression iff $\forall t \in I \tau(t) \models \varphi$. For each PORV, event, Boolean signal, or Boolean expression, φ , $\mathcal{I}_\varphi(\tau)$ is the set of all truth intervals of φ in τ . \square

For trace τ , $I_\tau = [L : R]$ is the time interval over which the trace is defined. In this context, the complement of a truth interval I , the false interval, is denoted $\bar{I} = \{t' | t' \in I_\tau, t' \notin I\}$.

In general, a sequence expression may be expressed as: $\phi_1 \theta_1 \phi_2 \theta_2 \dots \theta_{n-1} \phi_n$, where $\forall 1 \leq i \leq n \phi_i$ is a Boolean expression of propositions, PORVs and events, and $\forall 1 \leq j < n \theta_j$ is a list of sequence operators of the form $\# [a:b]$, $[*a:b]$ and $[*a]$. If θ_j contains a sequence of operators, they are always applied to ϕ_j from left to right. The following definitions

describe the interval arithmetic interpretation for sequence expressions.

Definition 10. Interval Set $\mathcal{I}(\tau)$: For trace τ , $\mathcal{I}(\tau)$ is choice of truth intervals, $\langle I_{P_1}, I_{P_2}, \dots, I_{P_k} \rangle$, $I_{P_i} \in \mathcal{I}_{P_i}(\tau), \forall P_i \in \mathbb{P}_\varphi$, where \mathbb{P}_φ is the set of all Boolean propositions, PORVs and events defined in the assertion φ . For ease of use $\mathcal{I}_P(\tau) = I_P \in \mathcal{I}(\tau)$. \square

The truth interval for a sequence expression is viewed as having two contexts, a *begin match* and an *end match* context, as defined by Definitions 5 and 6. For non-temporal artifacts such as PORVs, events and Boolean expressions, the two contexts evaluate to the same set of truth intervals. The computation of the begin and end match truth intervals for a sequence expression is computed recursively. As defined earlier, sequence operators are left associative. Brackets may also be used to describe sequences that break away from the default semantics. In general the computation is defined below.

Definition 11. Begin and End Match Truth Intervals: Given a choice $\mathcal{I}(\tau)$ of truth intervals, the end match interval, $\mathcal{M}_E(\varphi, \mathcal{I}(\tau))$, and begin match interval, $\mathcal{M}_B(\varphi, \mathcal{I}(\tau))$, for sequence expression φ is defined as follows:

$$\begin{aligned} \mathcal{M}_E(\varphi, \mathcal{I}(\tau)) &= \mathcal{M}_E(\hat{\varphi}, \mathcal{I}(\tau)) \cap \mathcal{I}_P(\tau), \varphi \equiv \hat{\varphi} \wedge P \\ &= \mathcal{M}_E(\hat{\varphi}, \mathcal{I}(\tau)) \cup \mathcal{I}_P(\tau), \varphi \equiv \hat{\varphi} \vee P \\ &= [l(\mathcal{M}_E(\varphi_1, \mathcal{I}(\tau))) + a : r(\mathcal{M}_E(\varphi_1, \mathcal{I}(\tau)))], \varphi \equiv \{\varphi_1\}[*a] \\ &= (\mathcal{M}_E(\varphi_1, \mathcal{I}(\tau)) \oplus [a : b]) \cap \mathcal{M}_E(\varphi_2, \mathcal{I}(\tau)), \varphi \equiv \varphi_1 \# [a:b] \varphi_2 \\ &= \mathcal{M}_E(\{\varphi_1\}[*a], \mathcal{I}(\tau)) \cap \mathcal{M}_E(\varphi_2, \mathcal{I}(\tau)), \varphi \equiv \{\varphi_1\}[*a:b] \varphi_2 \end{aligned}$$

$$\begin{aligned} \mathcal{M}_B(\varphi, \mathcal{I}(\tau)) &= \mathcal{M}_B(\hat{\varphi}, \mathcal{I}(\tau)) \cap \mathcal{I}_P(\tau), \varphi \equiv \hat{\varphi} \wedge P \\ &= \mathcal{M}_B(\hat{\varphi}, \mathcal{I}(\tau)) \cup \mathcal{I}_P(\tau), \varphi \equiv \hat{\varphi} \vee P \\ &= \mathcal{M}_B(\varphi_1, [l(\mathcal{M}_E(\varphi, \mathcal{I}(\tau))) : r(\mathcal{M}_E(\varphi, \mathcal{I}(\tau)))] - a), \varphi \equiv \varphi_1 [*a] \\ &= \mathcal{M}_B(\varphi_1, (\mathcal{M}_E(\varphi, \mathcal{I}(\tau)) \ominus [a : b]) \cap \mathcal{M}_E(\varphi_1, \mathcal{I}(\tau))), \\ &\quad \varphi \equiv \varphi_1 \# [a:b] \varphi_2 \text{ or } \varphi \equiv \{\varphi_1\}[*a:b] \varphi_2 \end{aligned}$$

\square

We recursively prove that the arithmetic defined in Definition 11 correctly computes the time points defined by assertion match semantics in Definitions 5 and 6.

The fundamental case, when φ is Boolean (BEXPR), is straightforward. The intervals of truth for the \wedge and \vee operations are respectively computed using the \cap and \cup set operations over intervals [17]. In these cases, the begin and end-matches are identical. For each of the four semantic rules, let $\tau(t) \models_e \varphi$, that is t is an end-match time point. We prove the arithmetic assuming the interval set $\mathcal{I}(\tau)$, a labelled set of truth intervals, one for each $P \in \mathbb{P}_\varphi$. Let $\mathcal{M}_E(\varphi, \mathcal{I}(\tau)) = [l : r]$ be a non-empty end-match interval. We use $\mathcal{D}(\cdot) : \mathbb{R}^+ \rightarrow I$, to generalize a time point to a time interval, in the context of a quantifier.

Theorem 1. The end-match intervals computed in Definition 11 correctly compute matches according to the semantics defined in Definition 5.

Proof. Let $\mathcal{M}_E(\varphi_1, \mathcal{I}(\tau)) = [l_1 : r_1]$ and $\mathcal{M}_E(\varphi_2, \mathcal{I}(\tau)) = [l_2 : r_2]$ be end-match truth intervals for φ_1 and φ_2 .

- $\varphi \equiv \{\varphi_1\}[*a]:$

$\mathcal{D}(t') = [0 : a]$ and $\mathcal{D}(t) = [l, r]$. Since $\tau(t - t') \models_e \varphi_1$, $\mathcal{D}(t - t') = [l_1, r_1]$. Hence, for a non-empty end-match interval for φ , $r_1 - l_1 \geq a$. Also, the earliest time point in $\mathcal{M}_E(\varphi, \mathbf{I}(\tau))$ is $l = \min_{(t-t') \in \mathcal{D}(\varphi_1)} (t - t') + \max_{t' \in [0:a]} (t') = l_1 + a$. While the latest time point is $r = \max_{(t-t') \in \mathcal{D}(\varphi_1)} (t - t') = r_1$.

- $\varphi \equiv \varphi_1 \text{ ##}[a:b] \varphi_2$: Using the domains of t and t' , we have, $\mathcal{D}(t) = [l_2 : r_2] \cap (\mathcal{D}(t') \oplus [a : b]) = [l_2 : r_2] \cap [l_1 + a : r_1 + b]$
- $\varphi \equiv \{\varphi_1\}[*a:b]\varphi_2$: As shown earlier, for a non-empty end-match, the right-hand side of the conjunction evaluated to the interval $[l_1 + a : r_1]$. Hence $\mathcal{D}(t) = [l_2, r_2] \cap [l_1 + a : r_1]$ \square .

Theorem 2. The begin-match intervals computed in Definition 11 correctly compute matches according to the semantics defined in Definition 6.

Proof. Let $\hat{t} \in \mathbb{B}(\tau, \varphi, t)$. We compute $\mathcal{D}(\hat{t})$ as follows:

- $\varphi \equiv \{\varphi_1\}[*a:] \mathcal{D}(t') = \mathcal{D}(t) - a = [l - a : r - a]$. So $\mathcal{D}(\hat{t}) = \mathcal{M}_B(\varphi_1, [l - a : r - a])$
- $\varphi \equiv \varphi_1 \text{ ##}[a:b] \varphi_2$ or $\{\varphi_1\}[*a:b]\varphi_2$: $\mathcal{D}(t) = [l : r]$, $\mathcal{D}(t'') = ([l : r] \ominus [a : b]) \cap \mathcal{M}_E(\varphi_1, \mathcal{D}(t))$, hence $\mathcal{D}(\hat{t}) = \mathcal{M}_B(\varphi_1, \mathcal{D}(t''))$ \square

Definition 12. Match Truth Interval for an Assertion : Given a choice of $\mathbf{I}(\tau)$ on a simulation trace τ , the match truth interval I_M for an assertion Φ is computed as follows:

- $\Phi : \varphi_1 \mapsto \varphi_2$, $I_M = \mathcal{M}_E(\varphi_1, \mathbf{I}(\tau)) \cap \mathcal{M}_B(\varphi_2, \mathbf{I}(\tau))$.
- $\Phi : \varphi_1 \mapsto \text{##}[a:b]\varphi_2$, $I_M = \mathcal{M}_E(\varphi_1, \mathbf{I}(\tau)) \cap (\mathcal{M}_B(\varphi_2, \mathbf{I}(\tau)) \ominus [a : b])$. \square

Theorem 3. The match truth interval computed in Definition 12 correctly computes matches according to the semantics defined in Definition 7.

Proof. Consider each statement in Definition 7.

- $\Phi : \varphi_1 \mapsto \varphi_2$, Follows directly from Definition 7
- $\Phi : \varphi_1 \mapsto \text{##}[a:b]\varphi_2$, $t \in I_M$ iff $t \models_e \varphi_1$ and $\exists t', t'', t' \in \mathcal{B}(\tau, \varphi_2, t'')$ for $t'' > t'$ and $t \in [t' - b : t' - a]$. Hence $I_M = \mathcal{M}_E(\varphi_1, \mathbf{I}(\tau)) \cap (\mathcal{M}_B(\varphi_2, \mathbf{I}(\tau)) \ominus [a : b])$ \square

A. Evaluating Property Matches

Consider the property describing the settling time of the output voltage V_{out} for an arbitrary circuit, as given below. Given the continuum of time when various predicates in the property are true, as shown in Figure 3, we describe how the truth of the property is evaluated using the definitions presented in the earlier sections.

```
property SettlingTime{;
  @+{V(Vout), 0.1*1.2} |-> ##[0.001:0.004]
  {V(Vout)>=0.95*1.2 && V(Vout)<=1.05*1.2}[*0.002];
endproperty
```

Let the event in the antecedent be denoted as E_1 and the PORV in the consequent is denoted as P_1 . We re-write the assertion as follows: $E_1 \mapsto \text{##}[0.001:0.004] P_1[*0.002]$, where

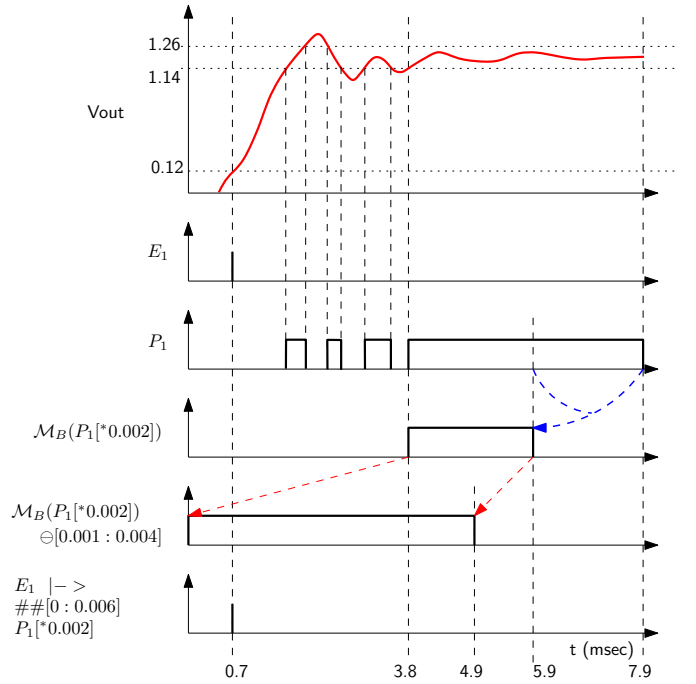


Fig. 3: Bottom-up evaluation of assertions.

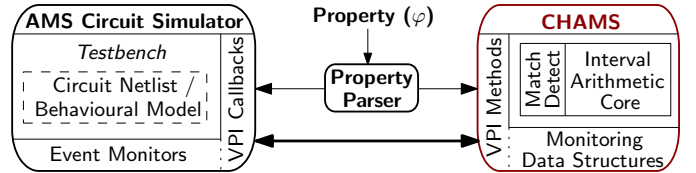


Fig. 4: CHAMS Tool Flow

$E_1 : @^+\{V(Vout), 0.1*1.2\}$, and

$P_1 : \{V(Vout) \geq 0.95*1.2 \ \&\& \ V(Vout) \leq 1.05*1.2\}$

Let for trace τ , $\mathcal{I}_{E_1}(\tau) = \langle [0.7ms : 0.7ms] \rangle$ and $\mathcal{I}_{P_1}(\tau) = \langle [1.6ms : 2.03ms], [2.41ms : 2.66ms], [3.04ms : 3.55ms], [3.8ms : 7.9ms] \rangle$. Figure 3 demonstrates the bottom-up approach used for computing the truth of the assertion. The begin and end of E_1 and P_1 are computed using Definition 11. Following the conditions mentioned in Section IV the assertion is computed to be true for the time interval $[0.7ms : 0.7ms]$.

V. THE CHECKER FOR AMS (CHAMS) TOOL

In this Section, we describe CHAMS, an online assertion checking tool for AMS. CHAMS works with off-the-shelf EDA tools to verify dense time AMS assertions during simulation. This paper extends CHAMS with recurrence operators. An overview of the tool is shown in Fig. 4.

A. Inputs

The inputs to the tool are a circuit netlist/behavioural model, its testbench and the assertions that need to be checked. Assertions are written in the syntax described in Section III. The assertion specification is analysed to automatically generate monitor codes (as explained in the following subsection) containing VPI-callback functions for the monitoring of events and PORVs affecting the truth of the assertion.

B. Monitor Generation

In order to maintain truth intervals and thereby compute the truth of the assertion, CHAMS generates Verilog-AMS

(VAMS) monitors and injects them into the VAMS testbench for the circuit. Note that it is not mandatory to have a VAMS testbench. In its absence monitor codes may also be placed in an independent module having access to circuit ports.

A monitor consists of standard VPI callbacks which send information about the state of the circuit to the checker CHAMS. CHAMS in turn maintains a data structure in which it updates the truth intervals for each sub-expressions bottom-up. It uses interval arithmetic to do this, and thereby computes the truth of the assertion. We demonstrate this using the property settling time described in the following example.

Example 4. Rising Sequence: *If the enable is asserted and the output voltage V_{out} crosses 10% of its rated value of 1.2V within 100 μ s, then thereafter, within 1ms to 4ms V_{out} must reach its steady state (explained below).*

```
property RisingSequence{;
  @+{enable} ##[0:0.0001] @+{V(Vout),0.1*1.2}
  |-> ##[0.001:0.004]
  {V(Vout)>=0.95*1.2 && V(Vout)<=1.05*1.2}[*0.002];
endproperty
```

Here *steady state* means, $V(V_{out})$ remains within $\pm 5\%$ of the rated voltage 1.2V for at least 2ms, and we express this requirement using the recurrence operator in the consequent of the above assertion. In time linear in the length of the property, CHAMS generates VAMS monitors, one for each event, and two for each predicate and Boolean expression. For the property above, the following monitors are generated.

```
assign flag_2 = flag_2_0 && flag_2_1;
always @(posedge enable)
  $checkerCall(0,0,$abstime);
always @(cross(Vout-0.1*1.2,+1,1e-9,1e-6)
  $checkerCall(0,1,$abstime);
always @(cross(Vout-0.95*1.2,+1,1e-9,1e-6)
  flag_2_0 = 1'b1;
always @(cross(Vout-0.95*1.2,-1,1e-9,1e-6)
  flag_2_0 = 1'b0;
always @(cross(Vout-1.05*1.2,+1,1e-9,1e-6)
  flag_2_1 = 1'b0;
always @(cross(Vout-1.05*1.2,-1,1e-9,1e-6)
  flag_2_1 = 1'b1;
always@(posedge flag_2)
  $updateTruthInterval(0,2,+1,$abstime);
always@(negedge flag_2)
  $updateTruthInterval(0,2,-1,$abstime);
```

The callbacks indicated in these monitors invoke the CHAMS assertion checker code whenever any event relevant to the assertion is detected during simulation. The first monitor issues a callback when *enable* goes high, the second monitor issues a callback when V_{out} crosses $0.1 * 1.2V$, and the third monitor examines the truth of the compound predicate $V(V_{out}) \geq 0.95 * 1.2 \ \&\& \ V(V_{out}) \leq 1.05 * 1.2$. \square

For each sub-expression, CHAMS maintains begin match (\mathcal{M}_B) and end match (\mathcal{M}_E) intervals. When the begin and end match for both the antecedent and the consequent are available, the truth of the assertion is evaluated, as given in Definition 12. In general, for a sequence $\varphi_1 \langle op \rangle \varphi_2$ containing

sub-sequences φ_1 and φ_2 , where $\langle op \rangle \in \{\#[a:b], [*a], [*a:b]\}$, treating $\langle op \rangle$ as left-associative, when φ_1 is found to be true, CHAMS monitors simulation progress for a time of at least b (or a , for $[*a]$) time units before deciding the match/fail of the sequence.

Definition 13. Depth of a Sequence Expression *The depth of a sequence expression φ , recursively defined as follows:*

$$\begin{aligned} \mathcal{D}(\psi) &= 0 \\ \mathcal{D}(\varphi [*a]) &= \mathcal{D}(\varphi) + |a| \\ \mathcal{D}(\varphi_1 \#[a:b]\varphi_2) &= \mathcal{D}(\varphi_1) + \mathcal{D}(\varphi_2) + b \\ \mathcal{D}(\varphi_1 [*a:b]\varphi_2) &= \mathcal{D}(\varphi_1) + \mathcal{D}(\varphi_2) + b \end{aligned}$$

where ψ is a Boolean expression. \square

For an assertion $\varphi_1 \mid \rightarrow \varphi_2$, evaluated over trace τ , to decide the truth of the assertion at time point t , where $t \in \mathcal{M}_E(s_1, \mathbf{I}(\tau))$, CHAMS allows simulation to progress upto $t + \mathcal{D}(\varphi_2)$.

VI. EMPIRICAL STUDIES

AMS assertion checking is relevant in two contexts, one in which the analog components are transistor level netlists, and one in which the analog components are replaced by behavioural models for accelerating simulation at the full-chip level. In order to study the overhead of our tool CHAMS, we have used two implementations of a Low Dropout Regulator (LDO) as test cases, namely a light-weight behavioural model written in Verilog-AMS, and an industry standard transistor level netlist of the same LDO. Several properties were coded in AMSAL, of which two are shown as illustrative examples.

Property 1. Settling time : *The settling time of the LDO should be less than 6 ms. The settling time is defined as the time taken by the system to settle down within $\pm 5\%$ of the rated voltage 3.2V and stay there for at least 2 ms.*

```
property SettlingTime{;
  @+{V(Vout),0.1*3.2} |-> ##[0:0.006]
  {V(Vout)>=0.95*3.2 && V(Vout)<=1.05*3.2}[*0.002];
endproperty
```

Property 2. Power Sequencing: *The power domain sourced by the LDO should not be enabled until the output of the LDO remains within $\pm 5\%$ of the rated voltage 3.2V for at least 10 ms. The enable signal for the power domain sourced by the LDO is called *en*.*

```
property Power_Sequencing{;
  ~{V(Vout)>3.15 && V(Vout)<3.25} |->
  ~{@+{en}}[*0.01];
endproperty
```

Table I shows the results of our empirical studies. In addition to the LDO, we also examined an industrial Buck Regulator netlist. All simulations were run using Cadence on a 2.33 GHz Intel-Xeon server with 32GB RAM. Column 3 represents the accuracy of the generated Verilog-AMS cross events used for monitoring the assertions. Both the Verilog-AMS BMOD and the transistor netlist of the circuit are run for the simulation time given in the table's second column. The standalone simulation time of the circuit, without assertion monitoring using CHAMS, is given in the fourth column. The fifth column shows the simulation time when the circuit is run

TABLE I: Results from Empirical Studies

DESIGN STATISTICS					
	#Nodes	#Transistors	#Resistors	#Capacitors	#Diodes
LDO Netlist	1434	336	1269	861	6
Buck Netlist	1787	2455	495	350	67
SIMULATION RESULTS					
Assertions	Simulation Time	Cross Event Accuracy (sec, Volt)	CPU Sim.Time		
			Ckt. only	Ckt. + CHAMS	% Overhead
LDO Behavioural Model in Verilog-AMS					
RisingSequence, Settling Time, Overshoot, Power_Seq	300ms	1e-4, 1e-3		11m 05s	7.78
		1e-6, 1e-4	10m 17s	11m 57s	16.21
		1e-9, 1e-6		13m 38s	32.57
LDO Transistor Level Netlist					
RisingSequence, Settling Time, Overshoot	40ms	1e-4, 1e-3		22m 46s	10.61
		1e-6, 1e-4	20m 35s	23m 42s	15.14
		1e-9, 1e-6		26m 7s	26.88
Buck Regulator Transistor Level Netlist					
RisingSequence, Settling Time, Overshoot	250 μ s	1e-4, 1e-3		3h 10m 14s	1.16
		1e-6, 1e-4	3h 8m 3s	3h 11m 07s	1.63
		1e-9, 1e-6		3h 17m 53s	5.23

with the assertion checker tool. A comparison of the last two columns indicates the overhead of the assertion monitoring.

The overhead shown in the results are largely due to the multiple VPI callbacks that have to be executed to accurately maintain the dense truth intervals for the PORVs. The major takeaway is that the assertion monitoring is online, hence a failure will be reported at the very instance when the assertion fails. Thus the designer can stop the simulating in-between whenever an assertion fails. Another appealing feature of CHAMS is that it is a completely automated tool which requires minimum user intervention.

VII. RELATED WORK

In our past work on AMS-LTL we extended temporal logic to express assertions for AMS [9] and further proposed adding property variables to the logic in AMS-LTL^L [10]. We also describe mechanisms for AMS property checking using standard simulators [11] and proposed using auxiliary state machines to aid in testing and verification flows. In Ref. [12] we discuss how dense-time debugging windows for property matches and failures may be chosen and refined. We also introduced a language a mechanism for the analysis of features [13], [14]. Features are quantitative properties for AMS systems that specify a set of behaviours over which analog measurements are computed. The language of features is developed to be similar to SVA, which is a widely used property specification language for discrete domains in the Semiconductor industry. SVA, itself, was primarily developed for digital systems and therefore is limited in its capacity to express AMS properties.

In work proposed by other groups, the proposed languages either do not support recurrences or lack the ability to monitor properties online with off-the-shelf AMS circuit simulators. In Ref. [15], [16], an assertion monitoring tool is presented which

analyzes output signals to compute the truth of assertions. The assertion language is developed by incorporating Timed Regular Expression (TRE) [19] into STL and suitable algorithms are presented for their match computation [20], [21]. Although TREs offer a rich set of operators, they lack support for *events*, *delays*, and restricts I in $\langle\varphi\rangle_I$ to have integer endpoints only. Authors in [22], [23] have designed an FJS-type online algorithm for computing whether a timed word matches a given specification, taken in the form of a timed automaton. In our approach, we use interval arithmetic and support a language with artifacts from circuits. Unlike other offerings, CHAMS can interact with industrial circuit simulators to guarantee accuracy of property matches. Furthermore, given the fact that we primarily target the semiconductor industry as our use-case, and the fact that SVA is already extensively used in the field, adoption of an SVA-like language would be easier for verification engineers. Other attempts have been made to express properties for AMS in PSL and evaluate these using modern analog simulators such as Spectre [18]. Assertions written on analog ports can be evaluated on a predefined digital clock, or at an analog event (such as a *cross* event in Verilog-AMS) introducing analog to the assertion checking process. However, dense-time temporal properties can not be expressed in the proposed format.

Property languages for AMS over dense-time, developed using SVA-like syntax do not presently support specification of recurrence. Deciding property matches and failures for properties involving recurrent behaviours, that hold continuously over a period of time, requires a different mechanism when compared with those without recurrence.

VIII. CONCLUSION

We believe that recurrence over continuous time is necessary to express many properties of AMS designs. Our proposal for recurrence operators in AMS addresses this requirement. We deliberately choose a syntax similar to SVA for ease of verification engineers conversant with SVA. However properties expressed in our language AMSAL are evaluated by our tool CHAMS using interval arithmetic as opposed to cycle based evaluation of SVA. The ability of CHAMS to work with standard commercial circuit simulators has facilitated its integration into the verification tool flow of multiple semiconductor companies.

REFERENCES

- [1] "SystemVerilog 3.1a Language Reference Manual"
- [2] "IEEE Standard for Property Specification Language (PSL)"
- [3] S. Konrad and B. H. Cheng, "Real-time specification patterns", *Proceedings of the 27th international conference on Software engineering. ACM, 2005*, pp. 372–381.
- [4] R. Alur and T. A. Henzinger, "Real-time logics: Complexity and expressiveness", *Information and Computation*, vol. 104, no. 1, pp. 35–77, 1993.
- [5] R. Alur, T. Feder, and T. A. Henzinger "The Benefits of Relaxing Punctuality", *J. ACM*, vol. 43, no. 1, pp. 116–146, Jan. 1996. [Online]. Available: <http://doi.acm.org/10.1145/227595.227602>
- [6] R. Alur and T. A. Henzinger, "A Really Temporal Logic", *J. ACM*, M, vol. 41, no. 1, pp. 181–203, Jan. 1994. [Online]. Available: <http://doi.acm.org/10.1145/174644.174651>
- [7] A. Ain and P. Dasgupta "Interpreting Local Variables in AMS Assertions during Simulation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018
- [8] R. Mukhopadhyay, S. K. Panda, P. Dasgupta, and J. Gough, "Instrumenting AMS Assertion Verification on Commercial Platforms", *ACM TODAES* 2009

- [9] S. Mukherjee, P. Dasgupta, S. Mukhopadhyay, S. Little, J. Havlicek, and S. Chandrasekaran, "Synchronizing AMS Assertions with AMS Simulation: From Theory to Practice", *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 21:1–21:47, Apr. 2009. [Online]. Available: <http://doi.acm.org/10.1145/1497561.1497564>
- [10] S. Mukherjee, P. Dasgupta, "Incorporating Local Variables in Mixed-Signal Assertions", *TENCON 2009 - 2009 IEEE Region 10 Conference*, Singapore, 2009, pp. 1-5.
- [11] S. Mukherjee, P. Dasgupta, "Assertion Aware Sampling Refinement: A Mixed-Signal Perspective", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 11, pp. 1772-1776, Nov. 2012.
- [12] S. Mukherjee, P. Dasgupta, "Computing Minimal Debugging Windows in Failure Traces of AMS Assertions", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31.11 (2012): 1776-1781.
- [13] A.Ain, A.A.B.D. Costa, and P. Dasgupta, "Feature Indented Assertions for Analog and Mixed-Signal Validation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.11 (2016): 1928-1941.
- [14] A.A.B.D Costa, G. Frehse, and P. Dasgupta, "Formal Feature Interpretation of Hybrid Systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018): 2474-2484.
- [15] D. Ničković, O. Lebeltel, O. Maler, T. Ferrère, D. Ulus, "AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic", *Tools and Algorithms for the Construction and Analysis of Systems 2018*
- [16] D. Ničković, O. Maler, "AMT: A property-based monitoring tool for analog systems", *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, Berlin, Heidelberg, 2007.
- [17] Maler O., Nickovic D. (2004) Monitoring Temporal Properties of Continuous Signals. In: Lakhnech Y., Yovine S. (eds) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. FTRTFT 2004, FORMATS 2004. Lecture Notes in Computer Science, vol 3253. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30206-3_12
- [18] P. Bhattacharya, D. O'Riordan, W. Hartong, "Mixed signal assertion-based verification", *Design and Verification Conference and Exhibition, 2011*
- [19] "Timed regular expressions", *Journal of ACM* 2002 vol. 49, no. 2, pp. 172–206, Mar.2002. [Online]. Available: <http://doi.acm.org/10.1145/506147.506151>
- [20] Ulus D., Ferrère T., Asarin E., Maler O. (2014) Timed Pattern Matching. In: Legay A., Bozga M. (eds) Formal Modeling and Analysis of Timed Systems. FORMATS 2014. Lecture Notes in Computer Science, vol 8711. Springer, Cham. https://doi.org/10.1007/978-3-319-10512-3_16
- [21] Ulus D., Ferrère T., Asarin E., Maler O. (2016) Online Timed Pattern Matching Using Derivatives. In: Chechik M., Raskin JF. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2016. Lecture Notes in Computer Science, vol 9636. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-49674-9_47
- [22] Waga, Masaki, Ichiro Hasuo, and Kohei Suenaga. "Efficient Online Timed Pattern Matching by Automata-Based Skipping." *Formal Modeling and Analysis of Timed Systems* (2017): 224–243
- [23] Waga M., André É. (2019) Online Parametric Timed Pattern Matching with Automata-Based Skipping. In: Badger J., Rozier K. (eds) NASA Formal Methods. NFM 2019. Lecture Notes in Computer Science, vol 11460. Springer, Cham. https://doi.org/10.1007/978-3-030-20652-9_26