Realtime Regular Expressions for Analog and Mixed-Signal Assertions

John Havlicek    Scott Little
Motivation

• Assertions are a key piece to industrial verification flows
• SVA and PSL are based upon discrete events
  - Work well for many digital circuits
  - Complex timing properties can be challenging
• Analog/mixed-signal (AMS) circuits are key SoC components
  - AMS blocks and interfaces are a disproportionate bug source
  - AMS properties involve relationships between events, event-based patterns, continuous time, and continuous quantities
• We need an assertion language with first class realtime support
• SVA and PSL are LTL-based discrete time temporal logics augmented with regular expressions

• Extending LTL for realtime has been well studied
  - TPTL, MTL, MITL, etc.

• Realtime regular expressions have been studied by Asarin, Caspi, and Maler

• What is left to be done?
• Previous work discusses discrete regular expressions or realtime regular expressions

• We provide a definition for realtime regular expressions that seamlessly intermingle with discrete regular expressions
  – Generalizes the SVA regular expressions
  – Enables writing complex mixed-signal regular expressions
• $A$ is the set of *analog variables*

• $D$ is the set of *discrete variables*

• A *state*, $s$ is an element of the set $\Sigma = \mathbb{R}^A \times \mathbb{B}^D$

• A *discrete trace* is a function $w : \{i \in \mathbb{N} : i \leq n - 1\} \rightarrow \Sigma$, where $0 \leq n \leq \infty$

• $b$ occurs in $s$ iff $s \models b$ (i.e., $b(s) = 1$)

• A *realtime trace* is a function $W : \mathbb{R}_{\geq 0} \rightarrow \Sigma$

• $b$ occurs in $W$ at $t$ iff $W(t) \models b$
• \( b \) is a boolean expression
• \( \kappa \) and \( \zeta \) are events
  – We require that events have no limit point in \( \mathbb{R} \)
• \( I, J \) denote bounded intervals in \( \mathbb{R} \) that may be open, closed, or half-open
• \( R \) is a realtime sequence
Semantics of digital sequences

\[ \sigma ::= @\(\kappa\)(b) \mid \sigma \#1 \sigma \mid \sigma \#0 \sigma \mid \sigma \text{ or } \sigma \]
\[ \mid \sigma \text{ intersect } \sigma \mid \sigma[^*0] \mid \sigma[+] \]

- Examples of discrete semantics
  - \( w \models_d @\(\kappa\)(b) \) iff \( |w| > 0 \) and \( b \) and \( \kappa \) occur at \( w|w|^{-1} \) and \( \kappa \) does not occur at any earlier position of \( w \).

- Examples of realtime semantics
  - \( W, l \models_r @\(\kappa\)(b) \) iff \( \{ t \in l : W(t) \models \kappa \} = \{ \sup l \} \) and \( W(\sup l) \models b \).

Examples of discrete semantics

- \( w \models_d @\(\kappa\)(b) \) iff \( |w| > 0 \) and \( b \) and \( \kappa \) occur at \( w|w|^{-1} \) and \( \kappa \) does not occur at any earlier position of \( w \).

Examples of realtime semantics

- \( W, l \models_r @\(\kappa\)(b) \) iff \( \{ t \in l : W(t) \models \kappa \} = \{ \sup l \} \) and \( W(\sup l) \models b \).
Prove that for digital sequences the realtime semantics are a faithful generalization of the discrete-time semantics

- Key feature that enables the intermingling of digital and realtime sequences
- Desire for this property shaped the realtime semantics and sequences
Realtime sequences

\[ R ::= @\(\kappa\)(b) \mid R \#1 R \mid R \#0 R \mid R \text{ or } R \]
\[ \mid R \text{ intersect } R \mid R[*0] \mid R[+] \]
\[ \mid b \mid b[*\alpha[+] : \beta[-]] \]

- Realtime (i.e., unclocked) boolean \((b)\)
  - \(W, I \models_r b\) iff there exists \(t\) such that \(I = \{t\}\) and \(W(t) \models b\)

- Boolean smear \((b[*\alpha[+] : \beta[-]])\)
  - \(W, I \models_r b[*\alpha : \beta]\) iff \(\alpha \leq |I| \leq \beta\) and \(W(t) \models b\) for all \(t \in I\)
    - \(\alpha\) denotes a non-negative rational constant
    - \(\beta\) denotes either a non-negative rational constant or the special symbol \($\), representing \(\infty\)
Derived realtime forms

- $b[*\alpha] \equiv b[*\alpha : \alpha]$ [exact-length smear]
- $b[\sim>1] \equiv !b[*0.0 : :]$ #1 $b$ [realtime goto]
- $R$ without $\odot(\kappa) \equiv R$ intersect $!\kappa[*0.0 : :]$ [sequence without an event].
- $R \#0 R' \equiv (R \#\#0 R')$ or $(R \#\#1 R')$ [flexible concatenation]
- $R \# [\alpha [+] : \beta [-]] R' \equiv R \#0 1[*\alpha [+] : \beta [-]] \#0 R'$ [concatenation with realtime delay]
- $R \# [\alpha] R' \equiv R \# [\alpha : \alpha] R'$ [concatenation with exact-length delay]
- $R[*] \equiv R[*0]$ or $R[+]$ [repetition]
- $R$ and $R' \equiv ((R \#0 1[*0.0 : :])$ intersect $R')$ or $(R$ intersect $(R' \#0 1[*0.0 : :]))$ [flexible intersection]
Endpoints and concatenation

• Allows the user to include, exclude, or not worry about endpoints
  - ##0 requires that it join a right-closed with a left-closed interval
  - ##1 joins a right-closed (resp., -open) interval with a left-open (resp., -closed) interval
  - Digital sequences and smear-free realtime sequences match over empty and right-closed intervals
  - Smear introduces the possibility of matching right-open intervals

• $@ (\kappa) (b) \#0 \ R$

• $@ (\kappa) (b) \#1 \ R$
Settling time of a DAC

• The 8-bit DAC input, \( \text{in} \), is latched on the rising edge of its clock, \( \text{clk} \). Settling time measurement begins when \( \text{in} \) equals \( \text{8'}\text{h}00 \) on the input for five cycles, followed by a change to \( \text{8'}\text{h}ff \) in the next clock cycle. The input is then required to remain \( \text{8'}\text{h}ff \) throughout the remainder of the measurement. The DAC output, \( \text{out} \), should then settle to \( 5 \text{ V} \pm 250 \text{ mV} \) after 50 ns of latching the \( \text{8'}\text{h}ff \) input. We understand settled to mean that the output remains within the specified voltage range for 25 ns after the initial 50 ns period has passed.

\[
@(\text{posedge } \text{clk})(\text{in} == \text{8'}\text{h}00)[*5] \#1 \\
@(\text{posedge } \text{clk})(\text{in} == \text{8'}\text{h}ff) \#0 \\
( (\text{in} == \text{8'}\text{h}ff)[*0.0:$] \text{ intersect} \\
1 \#[50.0n](\text{out} < 5.25 && \text{out} > 4.75)[*25.0n])
\]
Glitch detection (digital)

- Property: match positive glitches of 25 ns or less on a signal $a$
  
  @posedge $a$ @(posedge $s$) (@posedge $s$)($!a$)

- $s$ is a 1 ns sampling clock (it produces a posedge every 1 ns)
- Glitches $< 1$ ns may be missed

- Glitches $> 25$ ns and $< 27$ ns may be matched
Glitch detection (realtime)

@ (posedge a) (1) #0 (!a[~>1] intersect 1 [*0.0:25.0n])

- No sampling clock needed
- Time capture is accurate because it is not forced to ns boundaries
- Simulator not the user manages timing granularity
• A timed automaton $\mathcal{A}$ recognizes $R$ in the sense that for all $W$ and $I$, $W, I \models_r R$ iff $\mathcal{A}$ has an accepting run whose trace is satisfied by $W$ over the interval $I$
  - Each initial or final state is classified as inclusive or exclusive relating to the endpoint
  - The full trace of a run is restricted by inclusivity or exclusivity of the endpoints
Automata convenience features

- 0-time state: no time elapses while in the state (i.e., $\eta = 0$)
- + -time state: time elapses while in the state
  - Annotated with + in lower half of the state
- Ingresses and egresses
  - 0-time states
  - Label is 1
  - Closed circle indicates inclusive
  - Open circle indicates exclusive
• Provide an automaton for each primitive operator and rules to connect the automata to form sequences

• \( @(\kappa) (a) \#\#1 @((\zeta)) (b) \)

• Connection rule for \( R \#\#1 R' \)
  - inclusive ingress/egress must connect to an exclusive ingress/egress

• Other operators have similar rules
Relationships with timed regular expressions

- Provide a semantically faithful mapping from timed regular expressions of Asarin, Caspi, and Maler (ACM) into our realtime sequences to demonstrate they are no less expressive.
- Our realtime sequences are no more expressive than the timed regular expressions of ACM:
  - Given automata construction
  - Assuming a suitable translation conventions between different semantic models:
    - Time-event sequences of ACM allow discrete ordering of simultaneous events
    - We do not believe this has practical relevance and our realtime traces do not allow this
  - Definitive comparison requires additional nontrivial work and merits future consideration.
Conclusions

- There is a growing need for assertions with a first class notion of continuous time
- Proposed syntax and semantics for realtime sequences that generalize existing SVA
  - Enables seamless intermingling of discrete and realtime sequences
- Provide a basis for implementation with definition of automata recognizers
Future work

- Extend semantics to local variables and `first_match`
- Develop compatible semantics for SVA property operators
  - `|=>` is particularly problematic
- Investigate efficient implementations of realtime extensions
  - Concerns over the performance of these new forms have been raised by several EDA vendors
  - Consider $p |\rightarrow F \left[ 5 : 10n \right] q$
    - If $p$ is false no checking of $q$ is required for the next 10 ns
    - If $p$ is true then $q$ must be checked over the next 5-10 ns
    - Can this checking be done using only events and timers?
    - Can it be applied systematically across the entire realtime language?
- Analyze relationship between our realtime sequences and the timed regular expressions of ACM