Applying High-Level Synthesis for Synthesizing Hardware Runtime STL Monitors of Mission-Critical Properties

Konstantin Selyunin\textsuperscript{1}, Thang Nguyen\textsuperscript{2}
Andrei-Daniel Basa\textsuperscript{2}, Ezio Bartocci\textsuperscript{1}
Dejan Nickovic\textsuperscript{3}, Radu Grosu\textsuperscript{1}

\textsuperscript{1}Vienna University of Technology, Vienna, Austria
\textsuperscript{2}Infineon Technologies Austria AG, Villach, Austria
\textsuperscript{3}AIT Austrian Institute of Technology, Vienna, Austria
Challenges/Motivations

Data/Information Evaluation:
- Pos-SIM/Pos-MEAS
- Root-cause Analysis
- System Properties Monitoring
Challenges/Motivations

Data/Information Evaluation:
- Pos-SIM/Pos-MEAS
- Root-cause Analysis
- System Properties Monitoring

Specification Formalization using Temporal Logic:
- Formal rigorous semantics
- No ambiguities about the intended meaning of requirements
- Minimizes information losses due to different interpretation
Runtime Verification

Main features [Leu12]:

- Check correctness properties based on the actual execution of a software or hardware system
- Make sure that the implementation really meets its correctness properties (apart from the model)
- Use information available at runtime
- Monitor behavior or properties that have been statically proved or tested: employ RV as a redundancy mechanism in safety-critical systems
Main features [Leu12]:

- Check correctness properties based on the actual execution of a software or hardware system
- Make sure that the implementation really meets its correctness properties (apart from the model)
- Use information available at runtime
- Monitor behavior or properties that have been statically proved or tested: employ RV as a redundancy mechanism in safety-critical systems
Main features [Leu12]:

- Check correctness properties based on the actual execution of a software or hardware system
- Make sure that the implementation really meets its correctness properties (apart from the model)
- Use information available at runtime
- Monitor behavior or properties that have been statically proved or tested: employ RV as a redundancy mechanism in safety-critical systems
Runtime Verification

Main features [Leu12]:

- Check correctness properties based on the actual execution of a software or hardware system
- Make sure that the implementation really meets its correctness properties (apart from the model)
- Use information available at runtime
- Monitor behavior or properties that have been statically proved or tested: employ RV as a redundancy mechanism in safety-critical systems
You already know Temporal Logic (TL) when you say

- “It is always the case that the violation must not occur”
- “Eventually the system must recover”
- “The presentation until the coffee break”
You already know Temporal Logic (TL)

- when you say
  - “It is always the case that the violation must not occur”
  - “Eventually the system must recover”
  - “The presentation until the coffee break”

We already used Temporal Logic without knowing it!

TL is a *structured* way to reason about events on a time axis
Temporal Logic 101

You already know Temporal Logic (TL)

- when you say
  - “It is always the case that the violation must not occur”
  - “Eventually the system must recover”
  - “The presentation until the coffee break”

We already used Temporal Logic without knowing it!

TL is a structured way to reason about events on a time axis

Advantages of using Temporal Logic:

- removing ambiguity
- operating with mathematical objects
- allowing automation
Temporal logics: from LTL to STL\textsuperscript{1}

- Linear Temporal Logic (LTL)
  (A.Pnueli 1977)
  logical time, unbounded

\textsuperscript{1}thanks to A.Rodionova
Temporal logics: from LTL to STL\(^1\)

- Linear Temporal Logic (LTL)
  
  (A.Pnueli 1977)

  logical time, unbounded

- Metric Temporal Logic (MTL)
  

  continuous/discrete time, bounded, with punctual intervals

\(^1\) thanks to A.Rodionova
Temporal logics: from LTL to STL

- Linear Temporal Logic (LTL)  
  (A.Pnueli 1977)  
  logical time, unbounded

- Metric Temporal Logic (MTL)  
  continuous/discrete time, bounded, with punctual intervals

- Metric Interval Temporal Logic (MITL)  
  (R.Alur, T.Feder, T.Henzinger 1996)  
  continuous/discrete time, bounded, without punctual intervals

1 thanks to A.Rodionova
Temporal Logics 101

Temporal logics: from LTL to STL

- Linear Temporal Logic (LTL)
  
  (A.Pnueli 1977)
  
  logical time, unbounded

- Metric Temporal Logic (MTL)
  
  
  continuous/discrete time, bounded, with punctual intervals

- Metric Interval Temporal Logic (MITL)
  
  (R.Alur, T.Feder, T.Henzinger 1996)
  
  continuous/discrete time, bounded, without punctual intervals

- Signal Temporal Logic (STL)
  
  
  continuous/discrete time, bounded, comparison with reals

\(^1\)thanks to A.Rodionova
STL Spec: Ingredients

\[ \varphi := p \mid x \sim a \]

\[ \downarrow \]

comparison with real value

\[ \downarrow \]

predicate

\[ \downarrow \]

STL formula

\[ \varphi \]
STL Spec: Ingredients

\[ \varphi := p \mid x \sim a \mid \text{not } \varphi \mid \varphi_1 \text{ or } \varphi_2 \]

- logical operators
- comparison with real value
- predicate
- STL formula
STL Spec: Ingredients

\[ \varphi ::= p \mid x \sim a \mid \text{not } \varphi \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \text{ until}_{[t_1,t_2]} \varphi_2 \]

- Temporal operators
- Logical operators
- Comparison with real value
- Predicate

STL formula
STL Spec: Ingredients

\[ \varphi := p \mid x \sim a \mid \text{not } \varphi \mid \varphi_1 \text{ or } \varphi_2 \mid \varphi_1 \text{ until } [t_1, t_2] \varphi_2 \]

Temporal operators

- \textbf{eventually} \[ [t_1, t_2] \varphi = \text{true until} [t_1, t_2] \varphi \]
STL Spec: Ingredients

\[ \varphi ::= p \mid x \sim a \mid \text{not } \varphi \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \text{ until} [t_1, t_2] \varphi_2 \]

- **temporal operators**
- **logical operators**
- **comparison with real value**
- **predicate**

**STL formula**

**Temporal operators**

- \( \text{eventually} [t_1, t_2] \varphi = \text{true until} [t_1, t_2] \varphi \)
- \( \text{always} [t_1, t_2] \varphi = \text{not eventually} [t_1, t_2] \text{ not } \varphi \)
STL Spec: Ingredients

\[ \varphi := p \mid x \sim a \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \text{ until}[t_1, t_2] \varphi_2 \]

- **Predicate**
- **Comparison with real value**
- **Logical operators**
- **Temporal operators**

**Temporal operators**

- \( \text{eventually}[t_1, t_2] \varphi = \text{true until}[t_1, t_2] \varphi \)
- \( \text{always}[t_1, t_2] \varphi = \neg \text{eventually}[t_1, t_2] \neg \varphi \)
- \( \text{next} \varphi = \text{eventually}\{1\} \varphi = \text{always}\{1\} \varphi \)
STL: Past and Future

Evaluation of an STL formula on a time axis

Past
- Looking **backward** from $t_{start}$
- Always bounded (there is $t = 0$)

Future
- Looking **forward** from $t_{start}$
- Can be unbounded (future might be infinite)
STL Temporal Operators

Next:

\[(w, i) \models \text{next } \varphi \iff (w, i + 1) \models \varphi\]

The signal \(w\) satisfies an STL formula \(\text{next } \varphi\) at a time step \(i\) iff at a time step \(i + 1\) \(w\) satisfies \(\varphi\).
STL Temporal Operators

Next:

\[(w, i) \models \text{next } \varphi \iff (w, i + 1) \models \varphi\]

The signal \(w\) satisfies an STL formula \(\text{next } \varphi\) at a time step \(i\) iff at a time step \(i + 1\) \(w\) satisfies \(\varphi\).

Eventually:

\[(w, i) \models \text{eventually}_{[a,b]} \varphi \iff \exists j \in i + [a, b] \cap \mathbb{T} : (w, j) \models \varphi_2\]

The signal \(w\) satisfies \(\text{eventually}_{[a,b]} \varphi\) at a time step \(i\) if there exist a time point \(j\) in the interval \([a, b]\) where \(w\) satisfies \(\varphi\).
The signal $w$ satisfies an STL formula $\text{always}_{[a,b]} \varphi$ at a time step $i$ if for all time points $j$ in the interval $[a, b]$ $w$ satisfies $\varphi$. 

\[(w, i) \models \text{always}_{[a,b]} \varphi \quad \iff \forall j \in i + [a, b] \cap T : (w, j) \models \varphi_2\]
**STL Temporal Operators**

**Always:**

\[(w, i) \models \text{always}_{[a, b]} \varphi \quad \iff \quad \forall j \in i + [a, b] \cap T : (w, j) \models \varphi_2\]

The signal \(w\) satisfies an STL formula \(\text{always}_{[a, b]} \varphi\) at a time step \(i\) if for all time points \(j\) in the interval \([a, b]\) \(w\) satisfies \(\varphi\).

**Until:**

\[(w, i) \models \varphi_1 \text{until}_{[a, b]} \varphi_2 \quad \iff \quad \exists j \in (i + [a, b]) \cap T : (w, j) \models \varphi_2 \]

and \(\forall k : i < k < j, (w, k) \models \varphi_1\)

The signal \(w\) satisfies an STL formula \(\varphi_1 \text{until}_{[a, b]} \varphi_2\) at a time step \(i\) if there exists a time point \(j\) in the interval \([a, b]\) where \(\varphi_2\) holds and for all previous time steps \(\varphi_1\) holds.
Let's Do an Example

Req.123
Fall time should be maximum 1.0us
Let’s Do an Example

Req.123
Fall time should be maximum 1.0us
Let’s Do an Example

Req.123
Fall time should be maximum 1.0μs

Define regions

trans

high
Let’s Do an Example

Req.123
Fall time should be maximum 1.0us

Define regions

exit(high) →
trans until \([0,1.0\text{us}]\) enter(low)
Monitor Generation Flow

1. **System Invariant**
   - System Requirements
   - STL Formula Sets
   - Simplification
   - Pastification

2. **STL Validation**
   - Formalized System Properties in STL
   - STL Temporal Operators Modeling

3. **Offline MON Framework**
   - Pre-Verification Support
   - Formal System Evaluation

4. **Runtime Lab Evaluation**
   - Realization on a specific HW Platform
   - HW Runtime Monitors TopLevel
   - HW Runtime Monitors Generation
   - Behavioral Temporal Operator Models (C++)
   - Pos-SI Verification Support

**High-Level Synthesis**
Requirements

Behavioral Temporal Operator Models

HW Runtime Monitors TopLevel
HW Runtime Monitors Generation

Realization on a specific HW Platform

Runtime Lab Evaluation

Pos-SI Verification Support

High-Level Synthesis

Offline MON Framework

STL Validation

Formal System Evaluation

Formalized System Properties in STL

STL Temporal Operators Modeling

Behavioral Temporal Operator Models (C++)

AMT System Properties Monitoring Tool

Properties are Satisfied/Violated

Ref: Thang et. Dejan

Time Invariant System Requirements

STL Formula Sets

Pastification

Simplification

System Requirements

STL Formula Sets

STL Validation

Formal System Evaluation

STL Temporal Operators Modeling

Behavioral Temporal Operator Models (C++)

AMT System Properties Monitoring Tool

 Properties are Satisfied/Violated

Ref: Thang et. Dejan
Pastification & Simplification

Time Invariant
- System Requirements
  - STL Formula Sets
  - Pastification
    - Simplification

STL Validation
- Formalized System Properties in STL
  - STL Temporal Operators Modeling

Offline MON Framework
- Formal System Evaluation
- Presynthesis Support

Runtime Lab Evaluation
- HW Runtime Monitors TopLevel
- HW Runtime Monitors Generation
- Realization on a specific HW Platform
- Pos-SI Verification Support

High-Level Synthesis
 Behavioral Temporal Operator Models (C++)
Offline Monitoring

- Application Specific STL Hardware Runtime Monitor
- Runtime Lab Evaluation
- Formally Validated
- High-Level Synthesis
- Behavioral Temporal Operator Models (C++)
- Pres-SI Verification Support
STL Primitives

Formal Requirements (STL)

System Properties Monitoring Tool AMT

Properties are Satisfied / Violated

STL Validation

Formal System Evaluation

Offline MON Framework

STL Temporal Operators Modeling

Behavioral Temporal Operator Models (C++)
High Level Synthesis

- Time Invariant
- System Requirements
- STL Formula Sets
- Pastification
- Simplification
- STL Temporal Operators Modeling
- Formalized System Properties in STL
- Offline MON Framework
- Formal System Evaluation
- STL Validation
- HW Runtime Monitors TopLevel
- HW Runtime Monitors Generation
- Applied HLS to generate Application Specific STL Hardware Runtime Monitor
- Runtime Lab Evaluation
- Pre-SI Verification Support
- Realization on a specific HW Platform
- HW Runtime Monitors Generation
- HW Runtime Monitors TopLevel
- Behavioral Temporal Operator Models (C++)
- System Properties Monitoring Tool

Ref: Thang et. Dejan
Future temporal operators reason about events in the future

- every bounded future formula can be converted to past (so-called pastification)
- the verdict of a specification satisfaction is shifted in time
HDL Generation

- compare $HW$ & $SW$ implementations (HW specific data types)
- apply optimization directives (optimize for throughput or area: e.g. array partition, pipelining)
- synthesize the HDL
- co-simulate the synthesized code
- export IP
Case Study: Specification

Missile property: Specification

Detonation must not happen within 5 time units after rise of fire_en.

↑ launch_en → ◇[0:4] (↑ fire_en ∧ □[0:5] ¬detonation)

“When the missile received the launch enable signal, it must see the fire enable signal followed within the next four time points. After fire_en has arrived, no detonation is allowed for the next five time points.”
Case Study: Specification

**Missile property:** Specification

- Detonation must not happen within 5 time units after rise of fire_en
- $\uparrow \text{launch_en} \rightarrow \Diamond [0;4] (\uparrow \text{fire_en} \land \Box [0;5] \neg \text{detonation})$

"When the missile received the launch enable signal, it must see the fire enable signal followed within the next four time points. After fire en has arrived, no detonation is allowed for the next five time points."

**STL formalization:** Future formula

- Detonation must not happen within 5 time units after rise of fire_en
- $\uparrow \text{launch_en} \rightarrow \Diamond [0;4] (\uparrow \text{fire_en} \land \Box [0;5] \neg \text{detonation})$
Pastified property:

- Pastified specification

\[ \diamondsuit \{9\} \uparrow \ell \rightarrow \diamondsuit [0,4] \left( \diamondsuit \{5\} \uparrow f \land \Box [0,5] \neg d \right) \]
The takeaway message:

- From system level requirements to hardware monitors
- Signal Temporal Logic as a specification language
- High Level Synthesis for HDL generation
Martin Leucker.  
Teaching Runtime Verification.  