2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

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

Konstantin Selyunin[1], Thang Nguyen[2]
Andrei-Daniel Basa[2], Ezio Bartocci[1]
Dejan Nickovic[3], Radu Grosu[1]

[1]Vienna University of Technology, Vienna, Austria
[2]Infineon Technologies Austria AG, Villach, Austria
[3]AIT Austrian Institute of Technology, Vienna, Austria

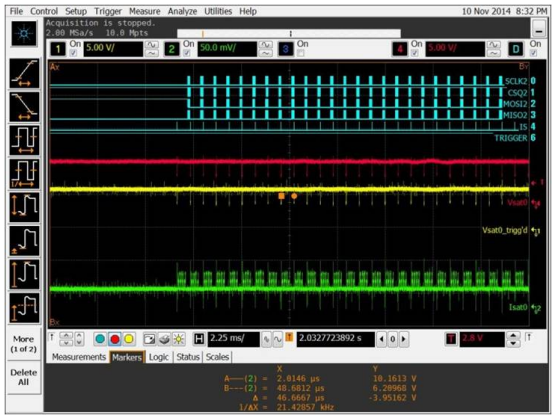
Challenges/Motivations

Data/Information Evaluation:

- Pos-SIM/Pos-MEAS
- Root-cause Analysis
- System Properties Monitoring

Challenges/Motivations

Data/Information Evaluation:

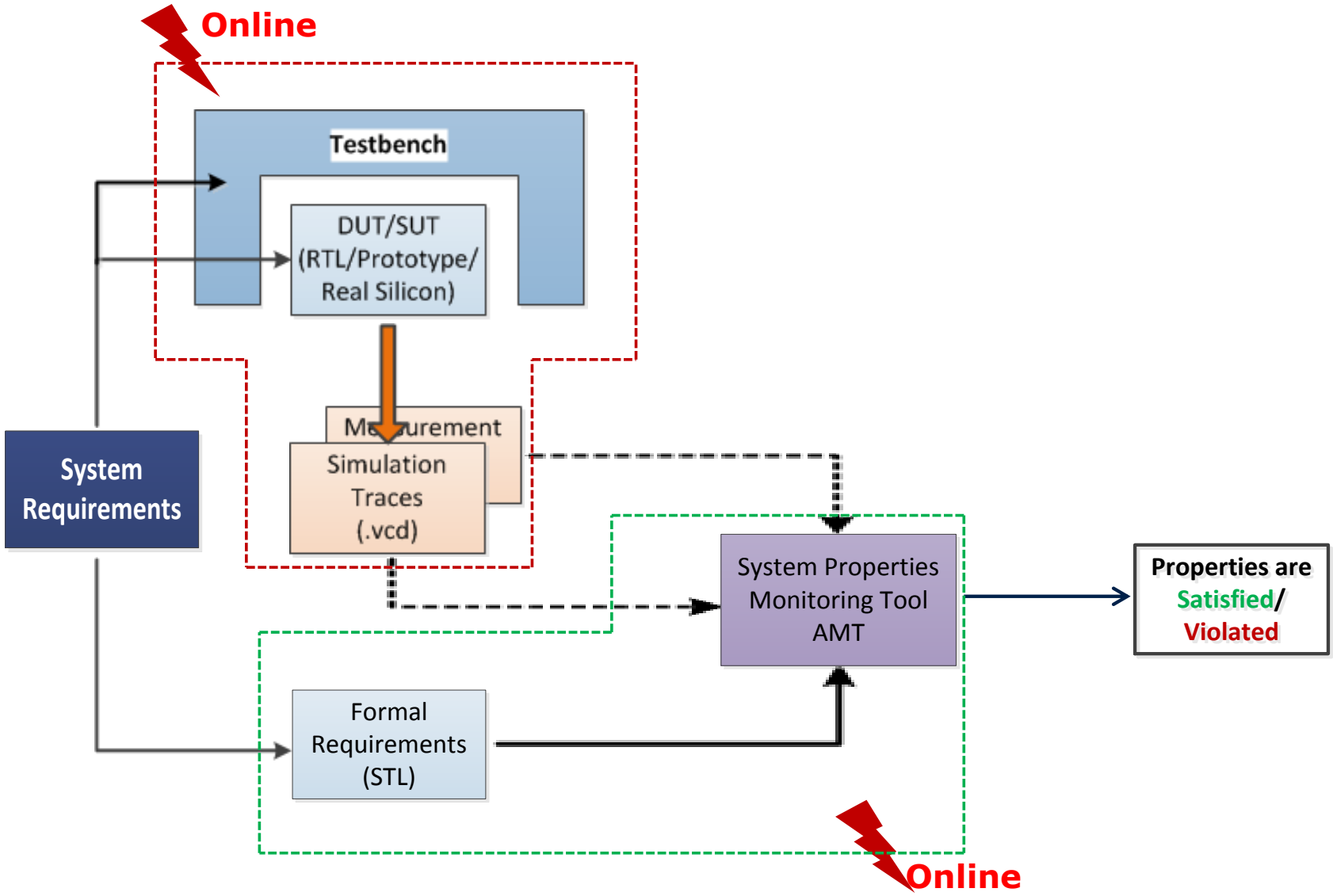


- Pos-SIM/Pos-MEAS
- Root-cause Analysis
- System Properties Monitoring

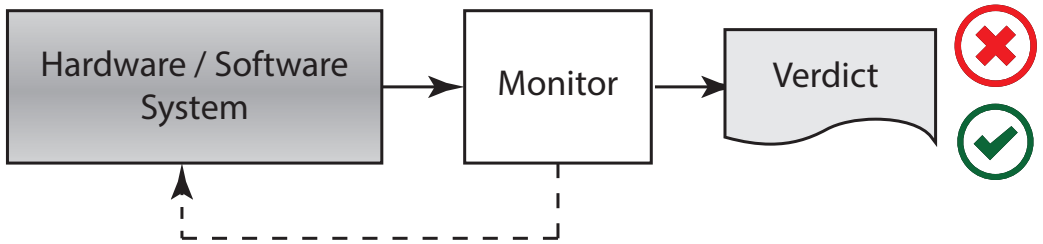
Specification Formalization using Temporal Logic:

- Formal rigorous semantics
- No ambiguities about the intended meaning of requirements
- Minimizes information losses due to different interpretation

Online Monitoring Framework Concept



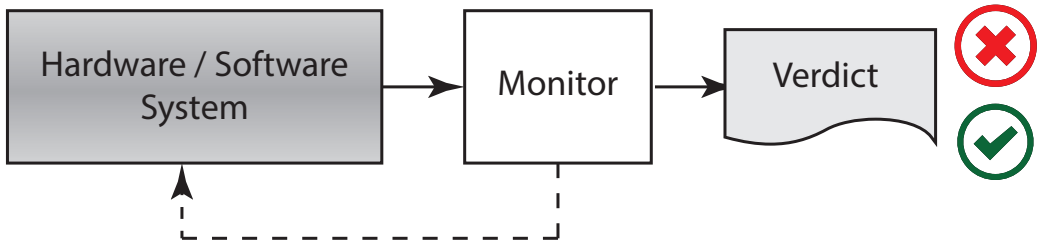
Runtime Verification



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

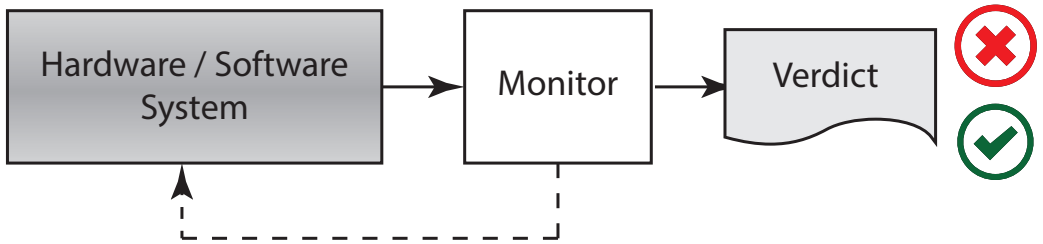
Runtime Verification



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

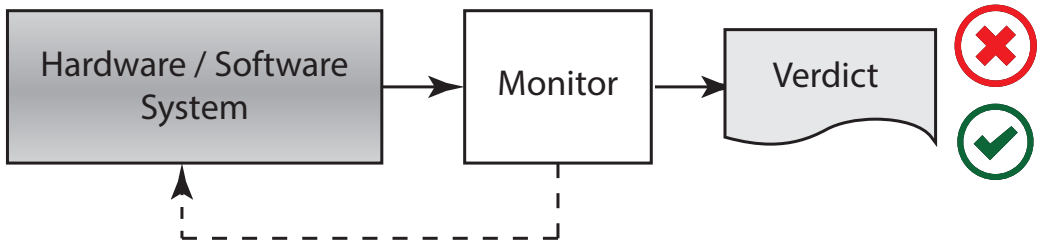
Runtime Verification



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

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

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"*

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

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

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 101

Temporal logics: from LTL to STL[1]

- Linear Temporal Logic (LTL)

  (A.Pnueli 1977)

  logical time, unbounded

[1] thanks to A.Rodionova

Temporal Logics 101

Temporal logics: from LTL to STL[1]

- Linear Temporal Logic (LTL)

  (A.Pnueli 1977)

  logical time, unbounded
- Metric Temporal Logic (MTL)

  (R.Koymans 1990, T.Henzinger 1993)

  continuous/discrete time, bounded, with punctual intervals

[1] thanks to A.Rodionova

Temporal Logics 101

Temporal logics: from LTL to STL[1]

- Linear Temporal Logic (LTL)

  (A.Pnueli 1977)

  logical time, unbounded
- Metric Temporal Logic (MTL)

  (R.Koymans 1990, T.Henzinger 1993)

  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[1]

- Linear Temporal Logic (LTL)

  (A.Pnueli 1977)

  logical time, unbounded
- Metric Temporal Logic (MTL)

  (R.Koymans 1990, T.Henzinger 1993)

  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)

  (O.Maler, D.Nickovic 2004)

  continuous/discrete time, bounded, comparison with reals

[1] thanks to A.Rodionova

STL Spec: Ingredients

$$\varphi := p \mid x \sim a \mid$$

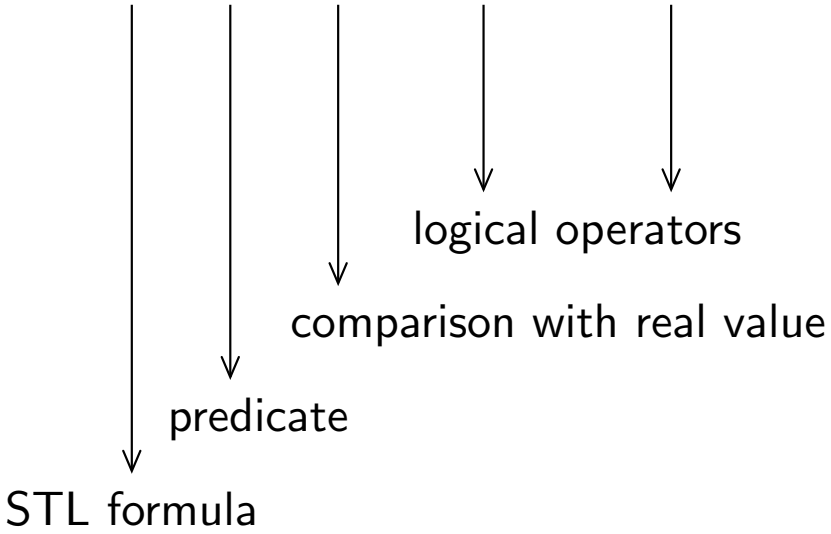
comparison with real value

predicate

STL formula

STL Spec: Ingredients

$$\varphi := p \mid x \sim a \mid \texttt{not}\ \varphi \mid \varphi_1\ \texttt{or}\ \varphi_2 \mid$$



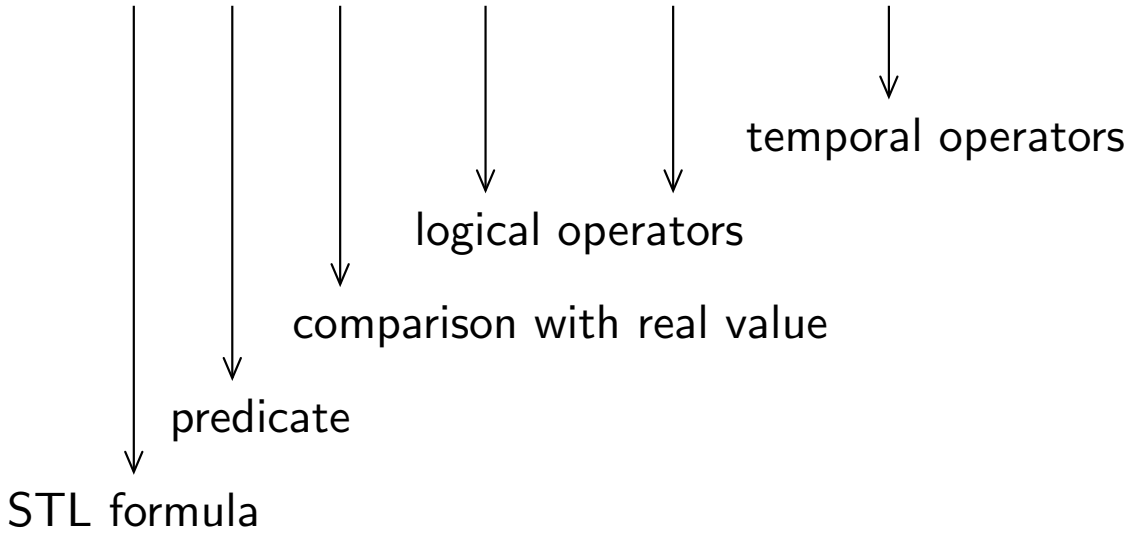
STL Spec: Ingredients

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

- temporal operators
- logical operators
- comparison with real value
- predicate
- STL formula

STL Spec: Ingredients

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

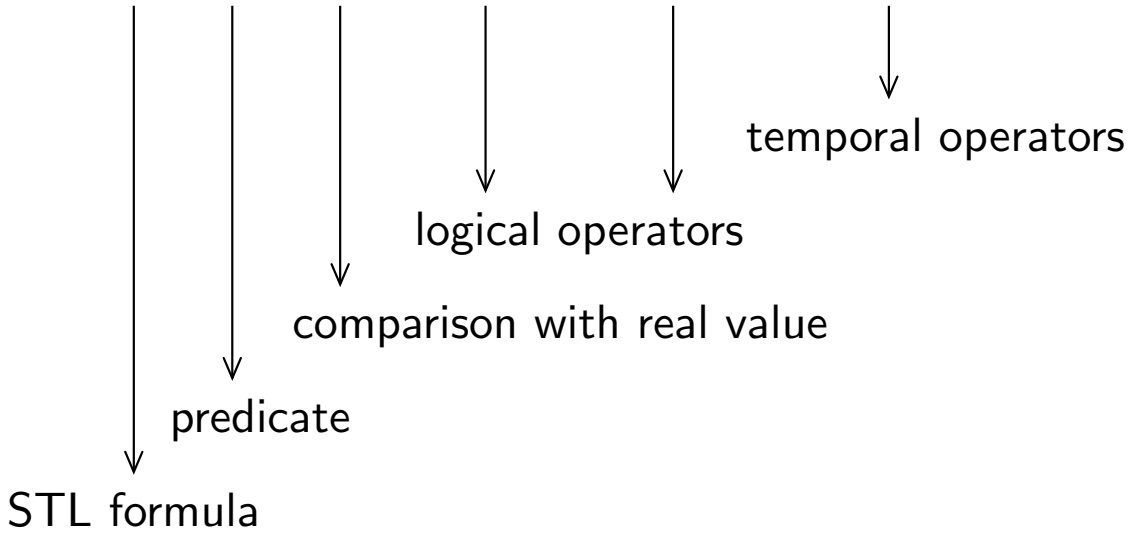


Temporal operators

- $\texttt{eventually}_{[t_1,t_2]}\ \varphi = \texttt{true}\ \texttt{until}_{[t_1,t_2]}\ \varphi$

STL Spec: Ingredients

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

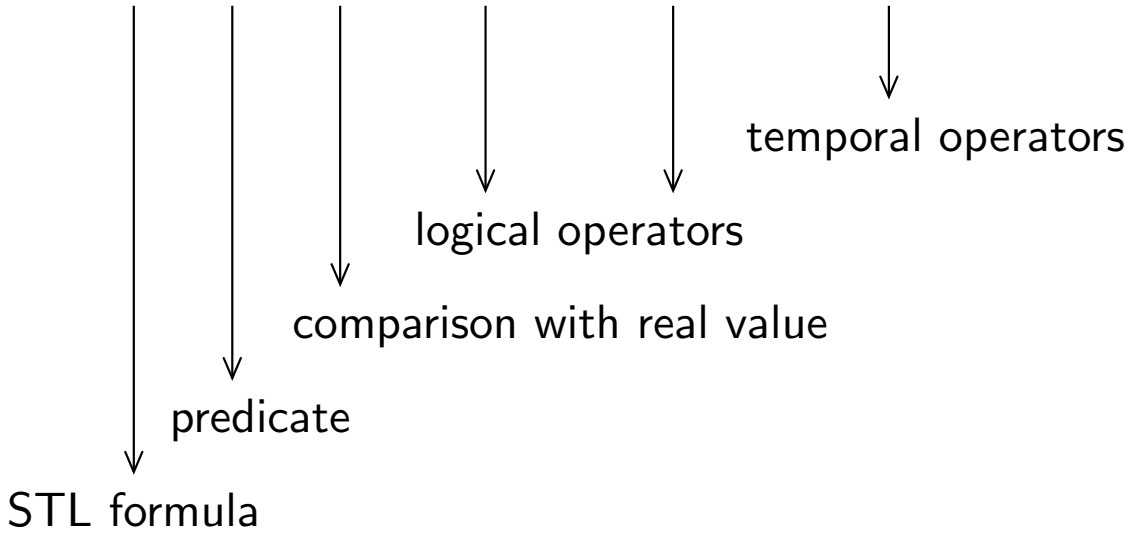


Temporal operators

- $\texttt{eventually}_{[t_1,t_2]}\ \varphi = \texttt{true}\ \texttt{until}_{[t_1,t_2]}\ \varphi$
- $\texttt{always}_{[t_1,t_2]}\ \varphi = \texttt{not}\ \texttt{eventually}_{[t_1,t_2]}\ \texttt{not}\ \varphi$

STL Spec: Ingredients

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

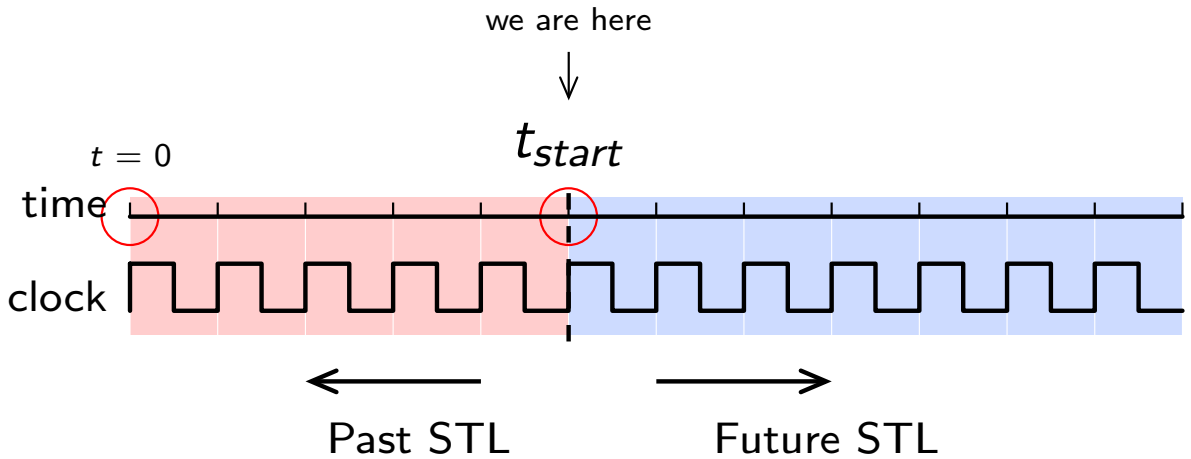


Temporal operators

- $\texttt{eventually}_{[t_1,t_2]}\ \varphi = \texttt{true}\ \texttt{until}_{[t_1,t_2]}\ \varphi$
- $\texttt{always}_{[t_1,t_2]}\ \varphi = \texttt{not}\ \texttt{eventually}_{[t_1,t_2]}\ \texttt{not}\ \varphi$
- $\texttt{next}\ \varphi = \texttt{eventually}_{\{1\}}\ \varphi = \texttt{always}_{\{1\}}\ \varphi$

STL: Past and Future

Evaluation of an STL formula on a time axis



Past STL: `previous`, $\texttt{once}_{[t_1,t_2]}$, $\texttt{historically}_{[t_1,t_2]}$, $\texttt{since}_{[t_1,t_2]}$

Future STL: `next`, $\texttt{eventually}_{[t_1,t_2]}$, $\texttt{always}_{[t_1,t_2]}$, $\texttt{until}_{[t_1,t_2]}$

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:

$$(w, i) \models \texttt{next}\,\varphi \quad \leftrightarrow \quad (w, i+1) \models \varphi$$

The signal w satisfies an STL formula $\texttt{next}\,\varphi$ at a time step i iff at a time step $i+1$ w satisfies φ.

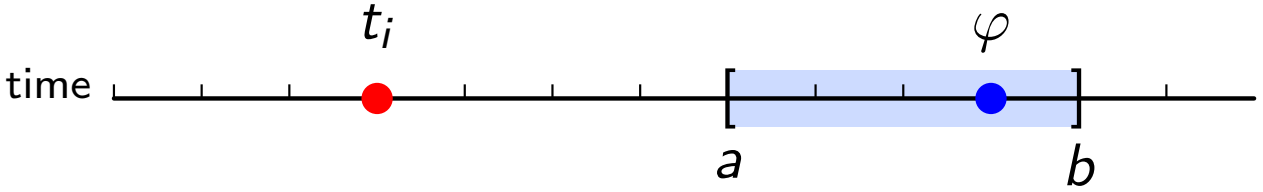
STL Temporal Operators

Next:

$$(w, i) \models \texttt{next}\,\varphi \qquad \leftrightarrow \qquad (w, i+1) \models \varphi$$

The signal w satisfies an STL formula $\texttt{next}\,\varphi$ at a time step i iff at a time step $i+1$ w satisfies φ.

Eventually:

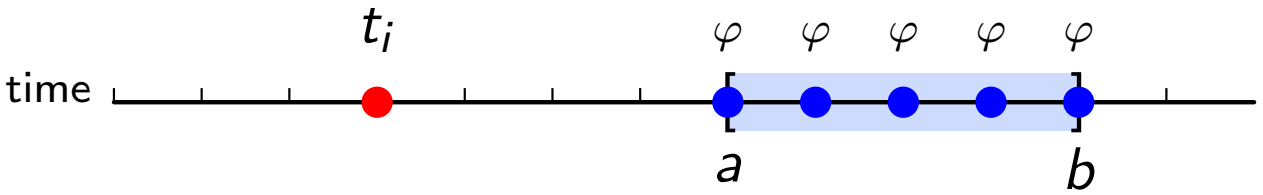


$$(w, i) \models \texttt{eventually}_{[a,b]}\varphi \qquad \leftrightarrow \exists j \in i + [a, b] \cap \mathbb{T} : (w, j) \models \varphi_2$$

The signal w satisfies $\texttt{eventually}_{[a,b]}\varphi$ at a time step i if there exist a time point j in the interval $[a, b]$ where w satisfies φ.

STL Temporal Operators

Always:

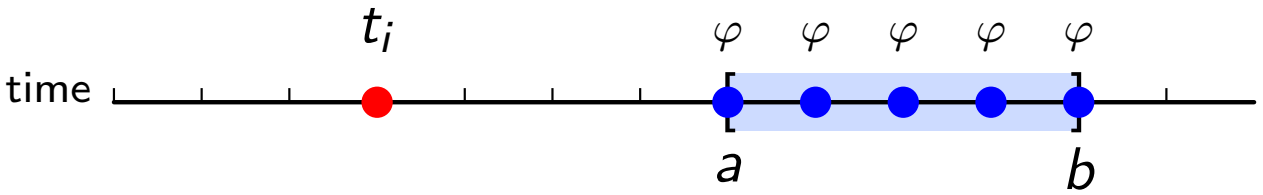


$$(w, i) \models \texttt{always}_{[a,b]}\varphi \quad \leftrightarrow \forall j \in i + [a, b] \cap \mathbb{T} : (w, j) \models \varphi_2$$

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

STL Temporal Operators

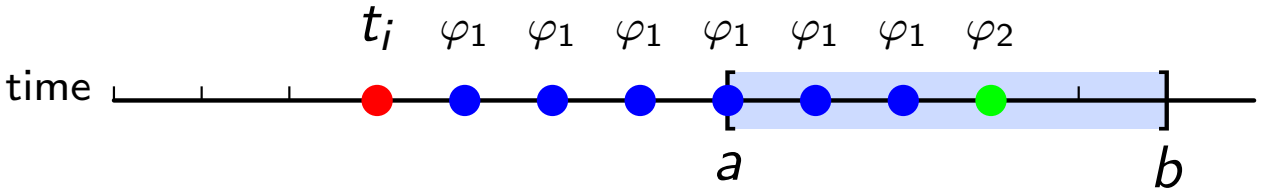
Always:



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

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

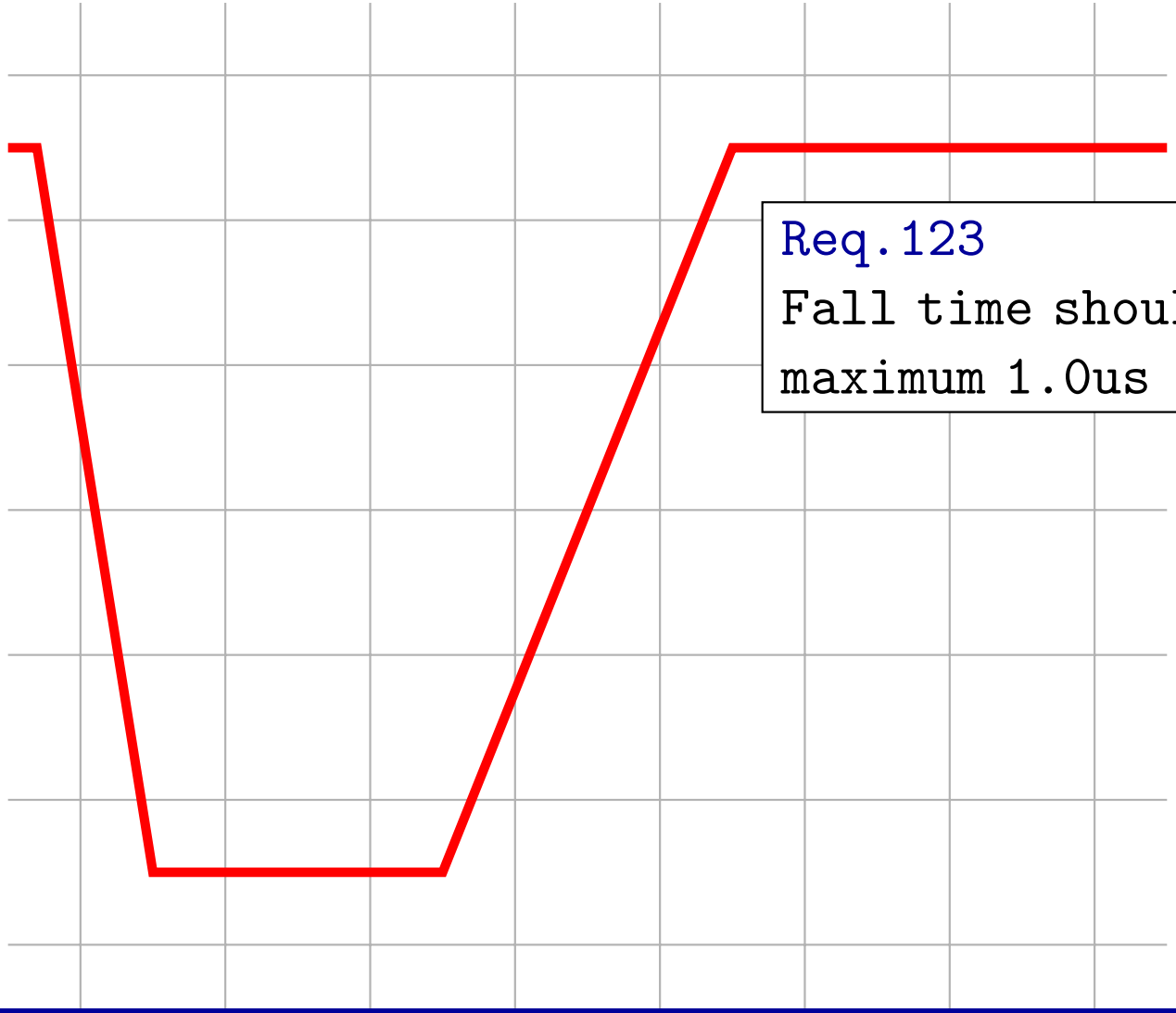
Until:



$$(w, i) \models \varphi_1 \texttt{until}_{[a,b]} \varphi_2 \quad \leftrightarrow \exists j \in (i + [a, b]) \cap \mathbb{T} : (w, j) \models \varphi_2$$
$$\text{and } \forall k : i < k < j, (w, k) \models \varphi_1$$

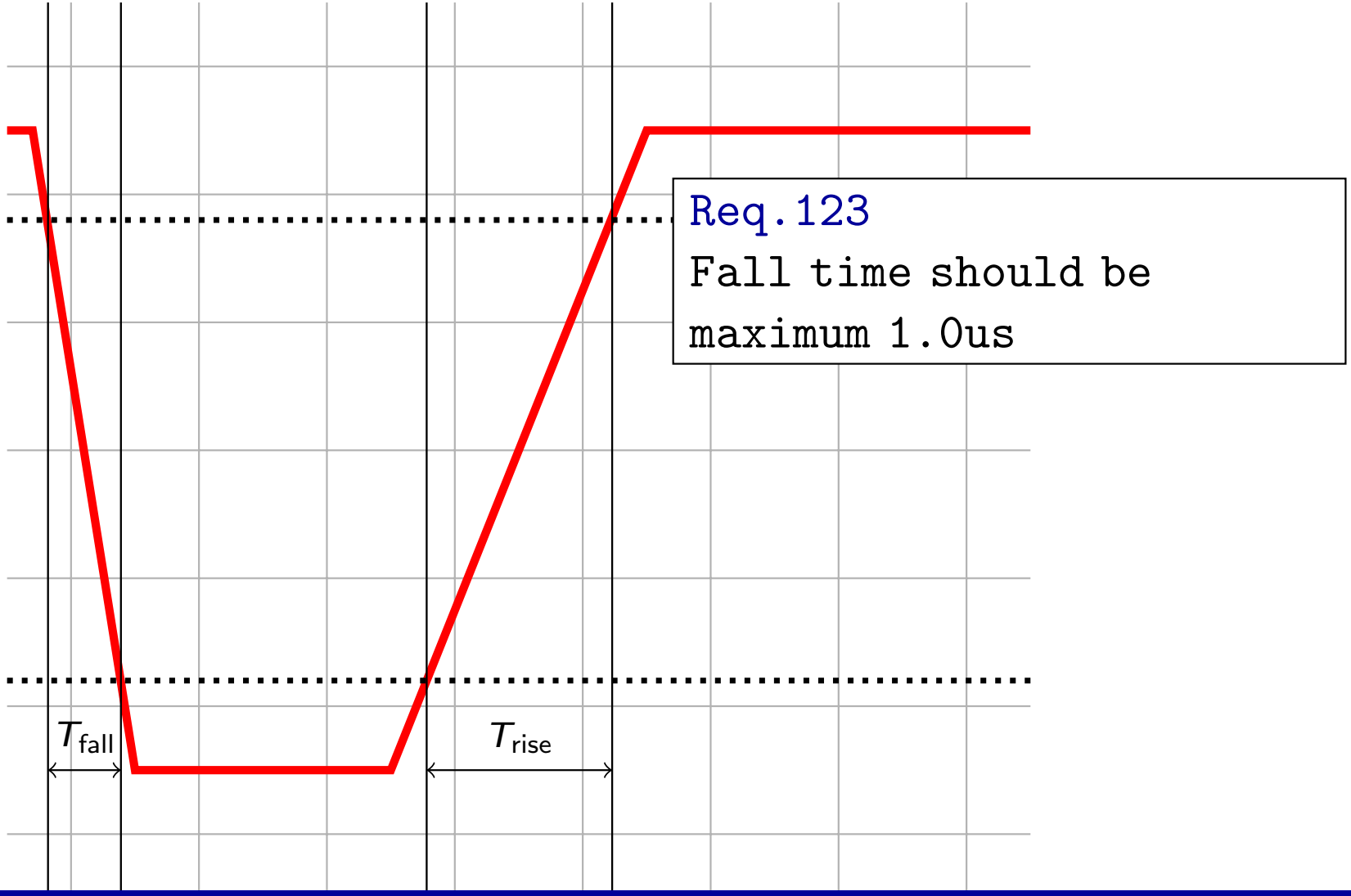
The signal w satisfies an STL formula $\varphi_1 \texttt{until}_{[a,b]} \varphi_2$ at a time step i if there exists a time point j in the interval $[a, b]$ where φ_2 holds and for all previous time steps φ_1 holds.

Let's Do an Example

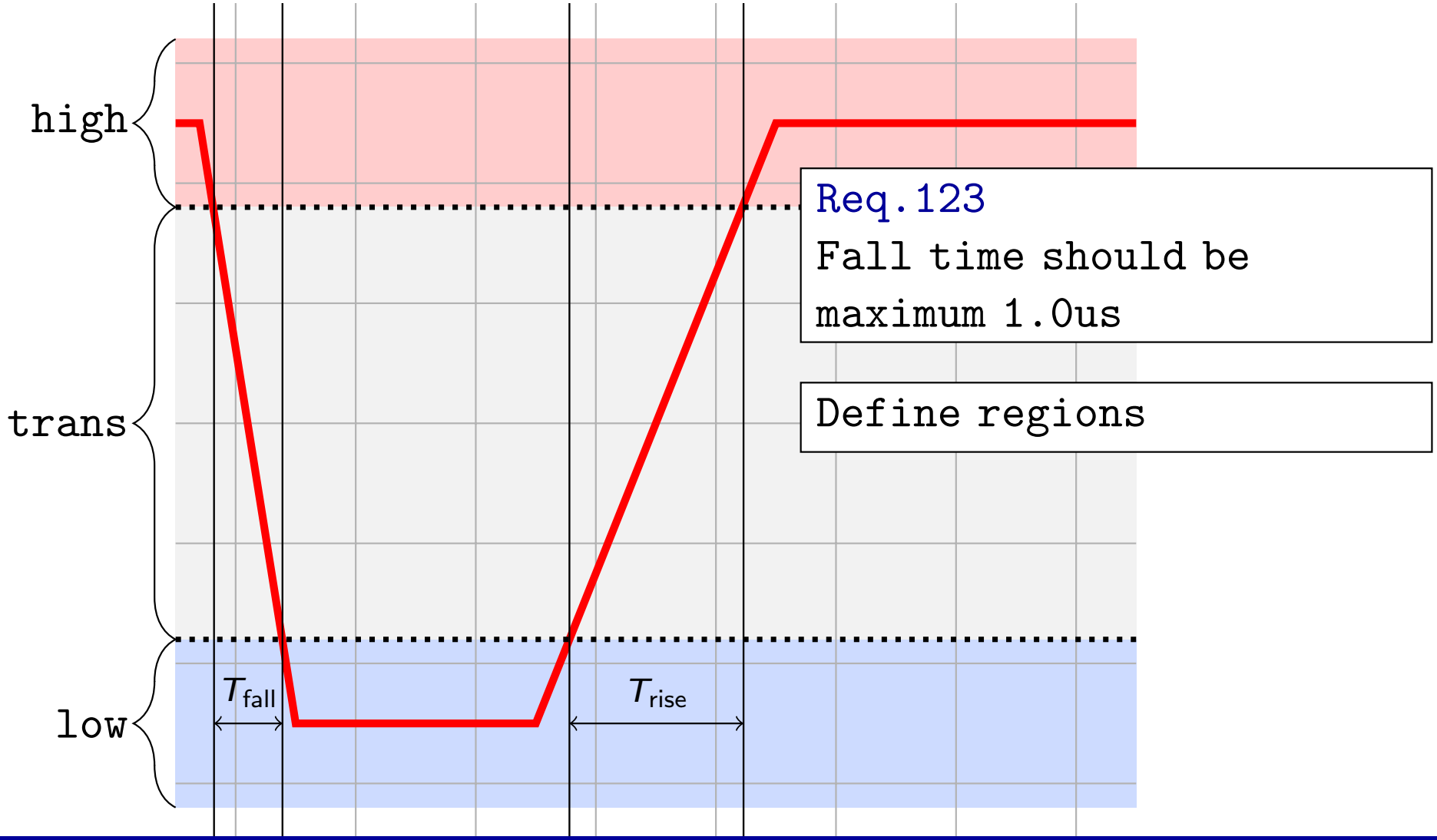


Req.123
Fall time should be
maximum 1.0us

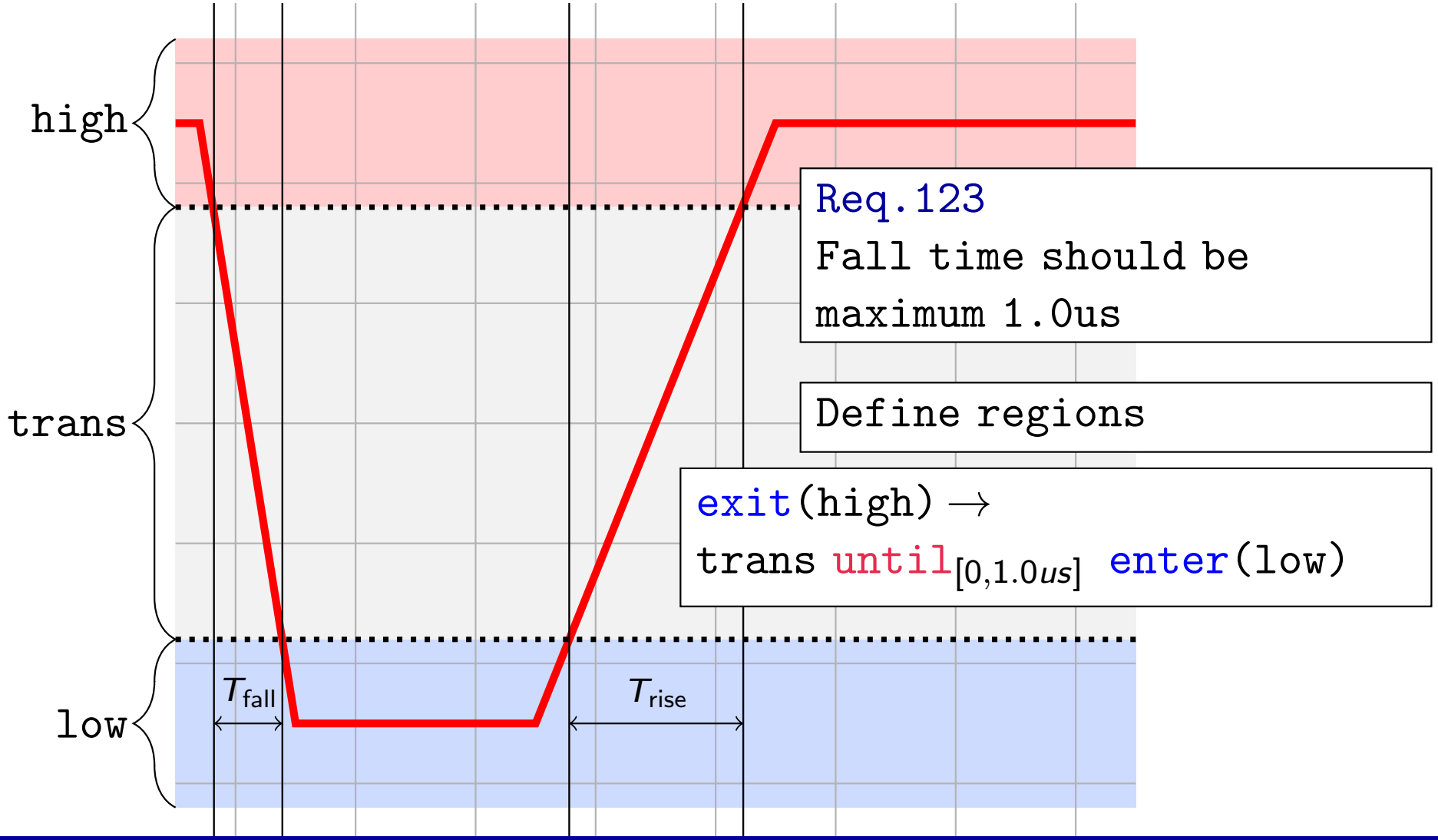
Let's Do an Example



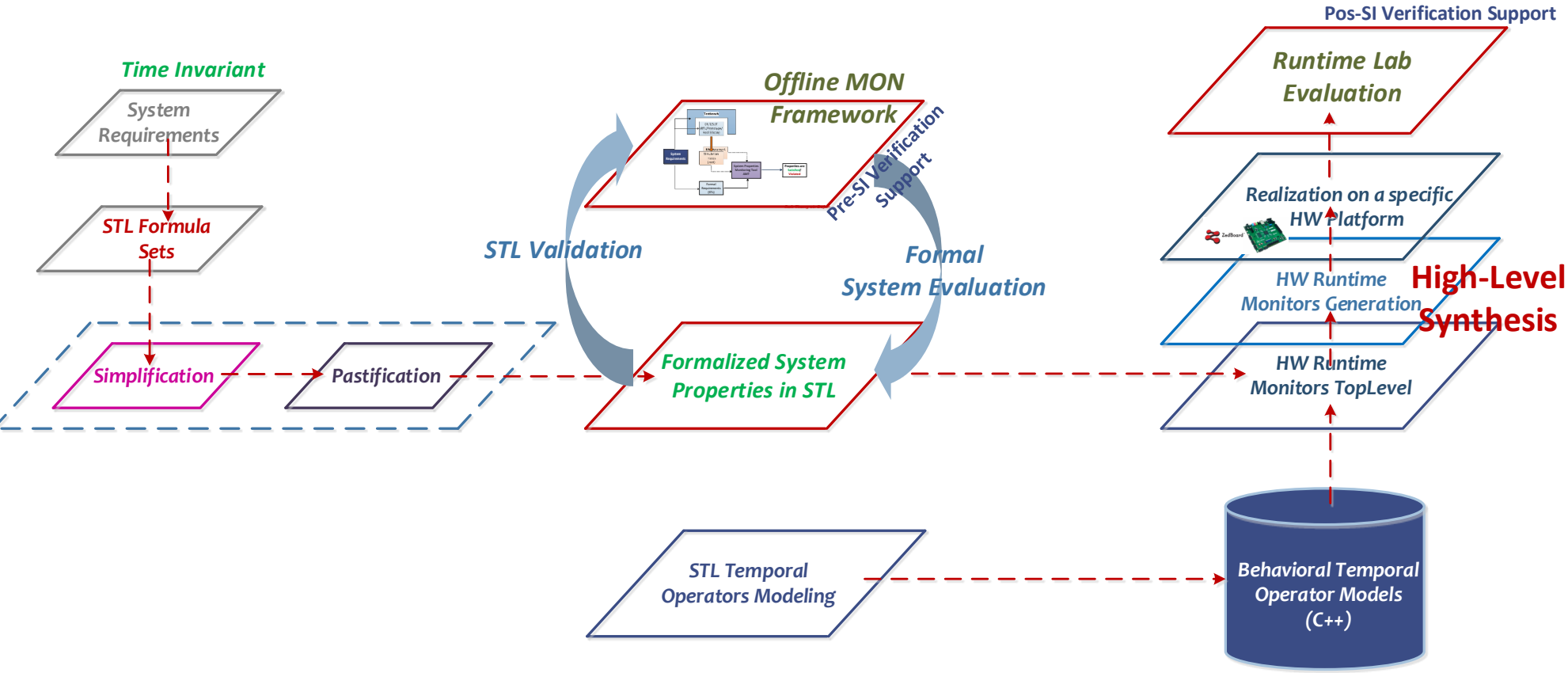
Let's Do an Example



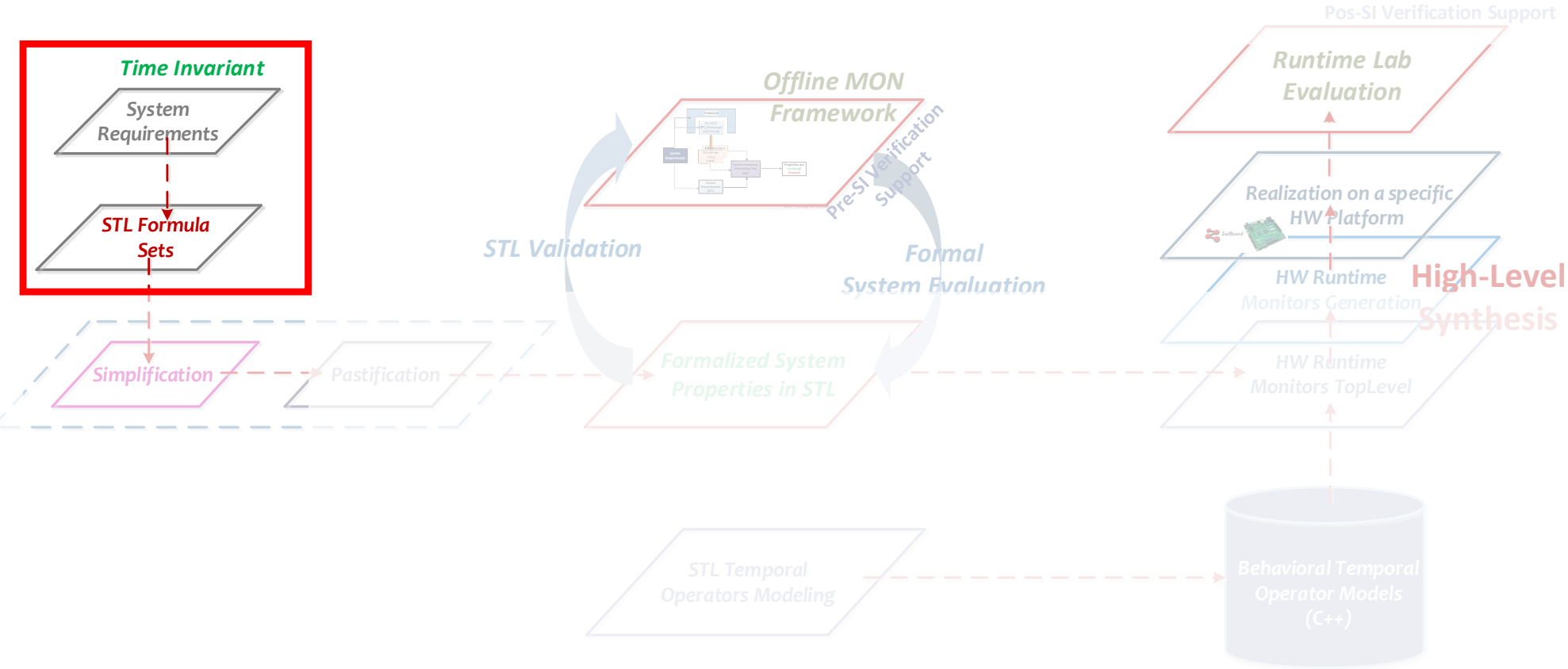
Let's Do an Example



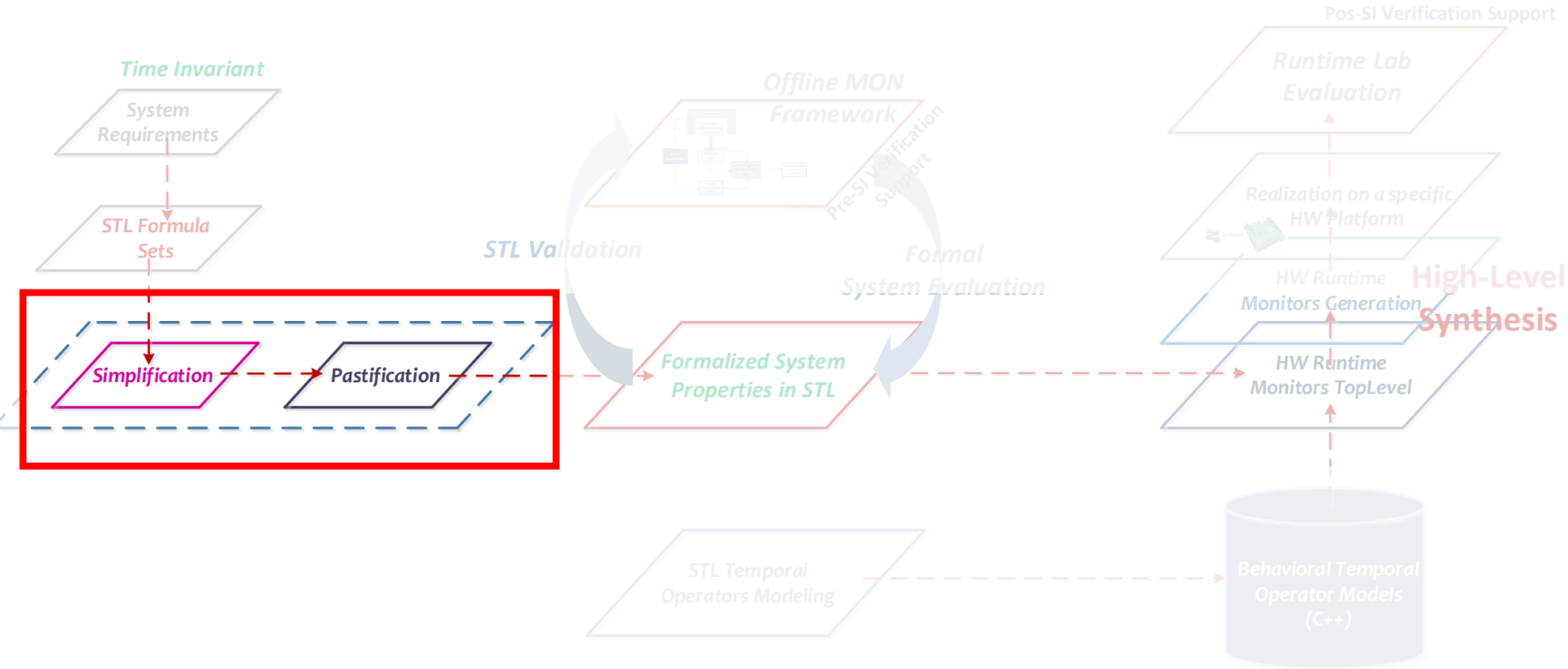
Monitor Generation Flow



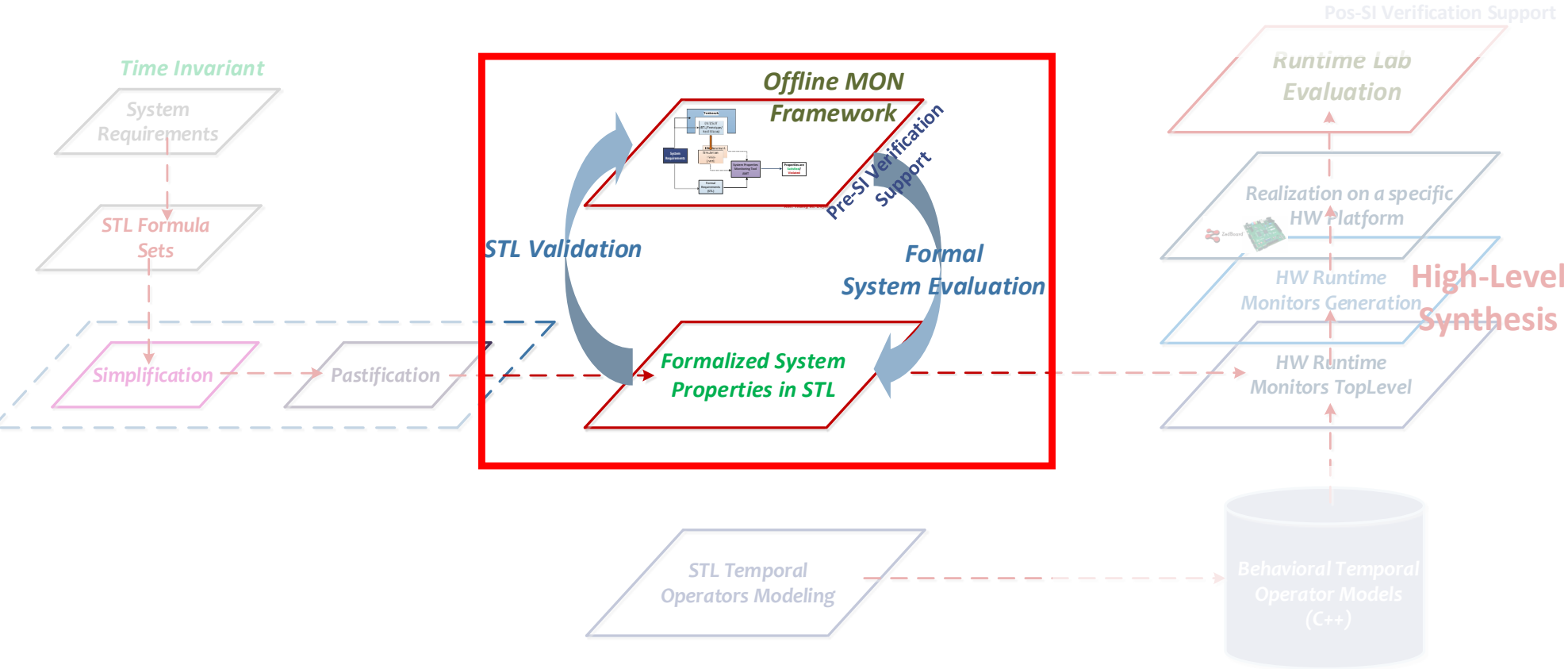
Requirements



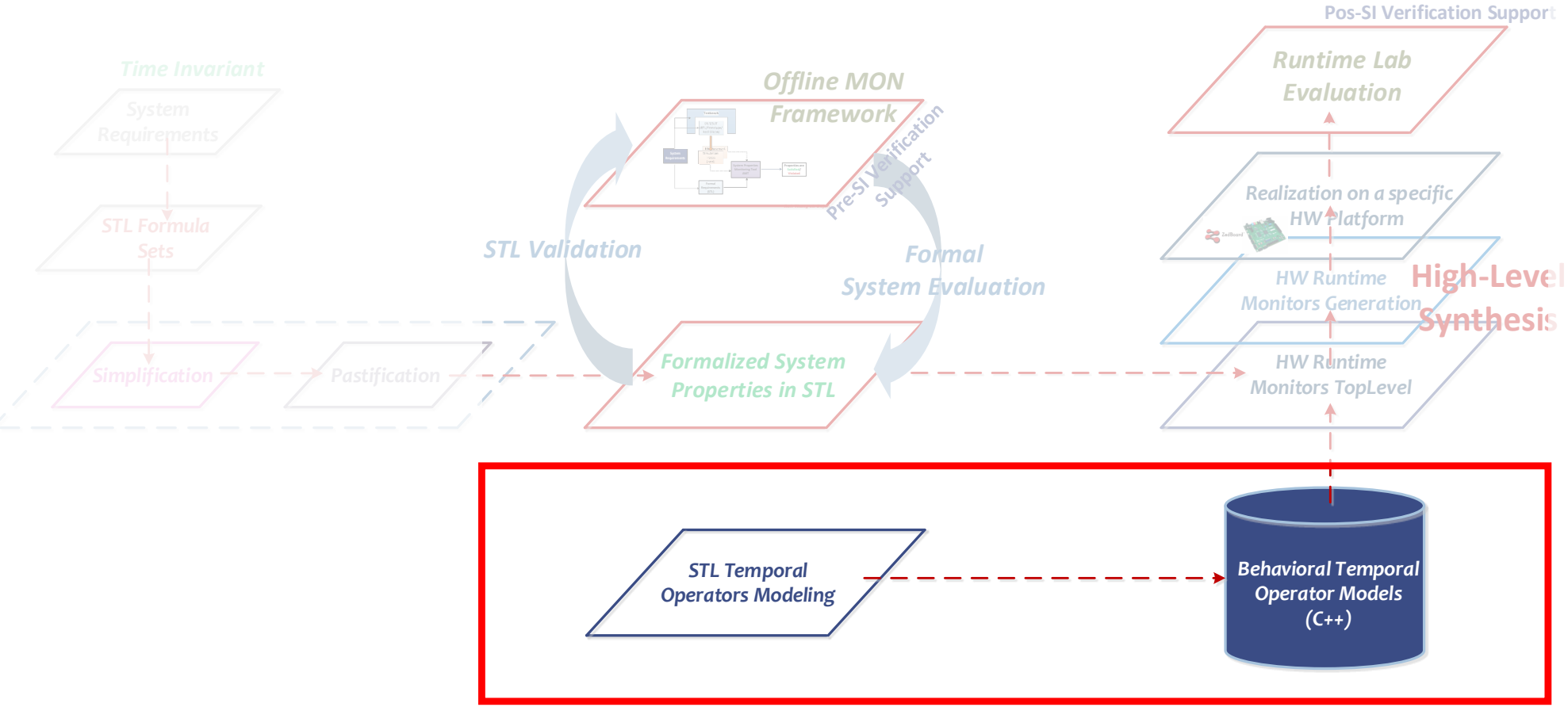
Pastification & Simplification



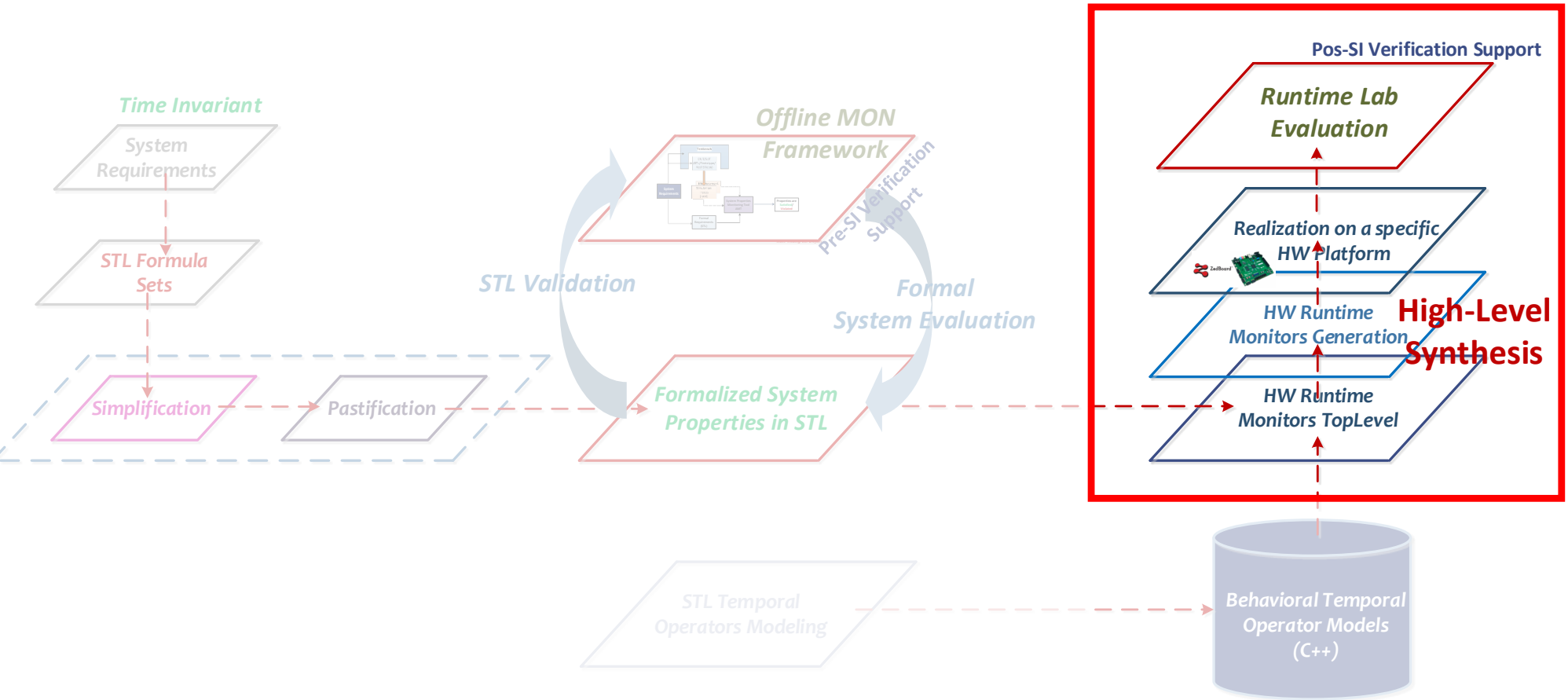
Offline Monitoring



STL Primitives



High Level Synthesis



STL: from Future to Past

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

High Level Synthesis

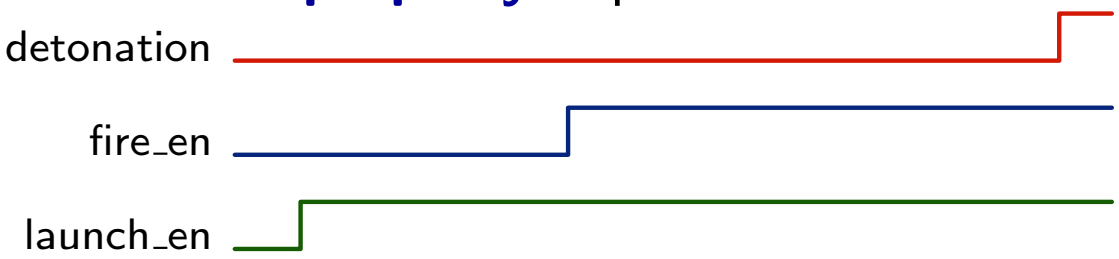
HDL Generation

- 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

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Case Study: Specification

Missile property: Specification



"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."

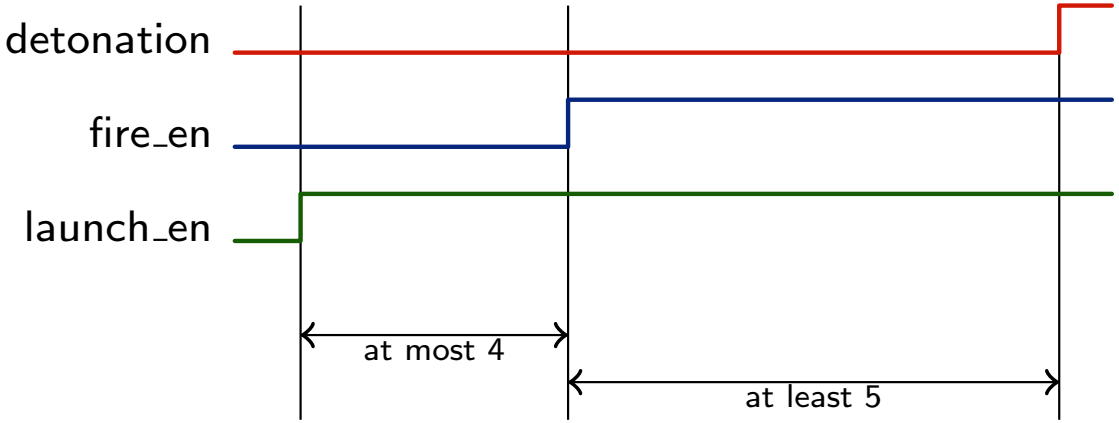
STL formalization: Future formula

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

$$\uparrow \text{launch\_en} \rightarrow \Diamond_{[0;4]} \left(\uparrow \text{fire\_en} \wedge \Box_{[0;5]} \neg \text{detonation} \right)$$

Case Study: Specification

Missile property: Specification



"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

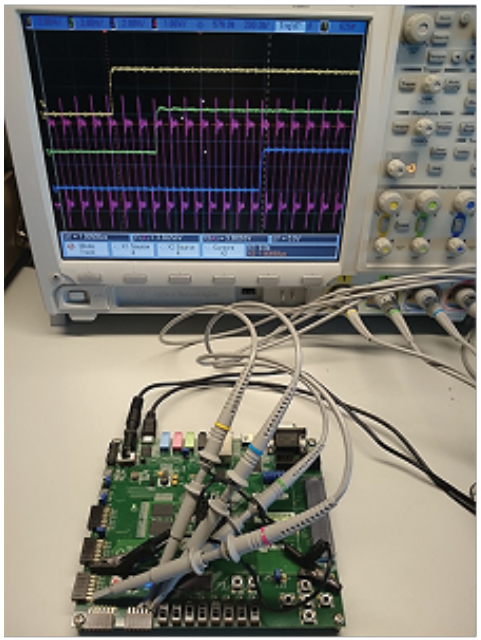
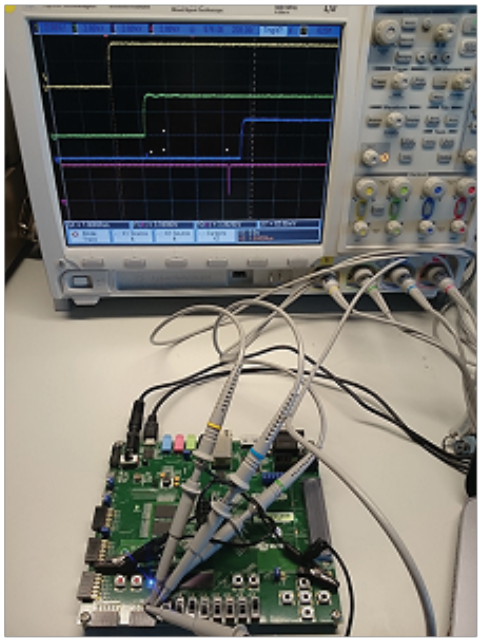
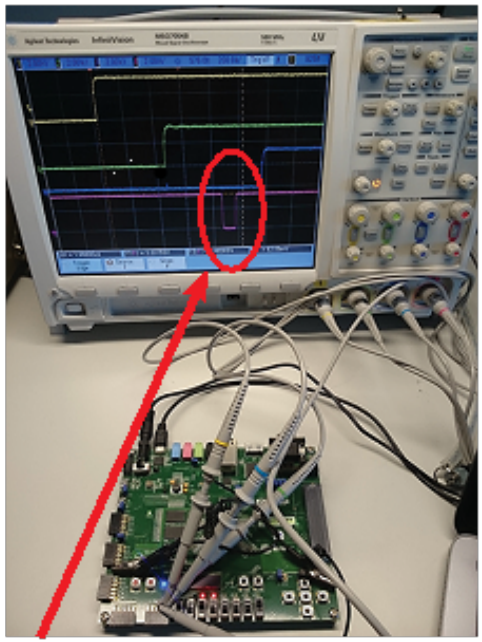
$$\uparrow \text{launch\_en} \rightarrow \Diamond_{[0;4]} \left(\uparrow \text{fire\_en} \wedge \Box_{[0;5]} \neg \text{detonation} \right)$$

Monitor Generation

Pastified property:

- Pastified specification

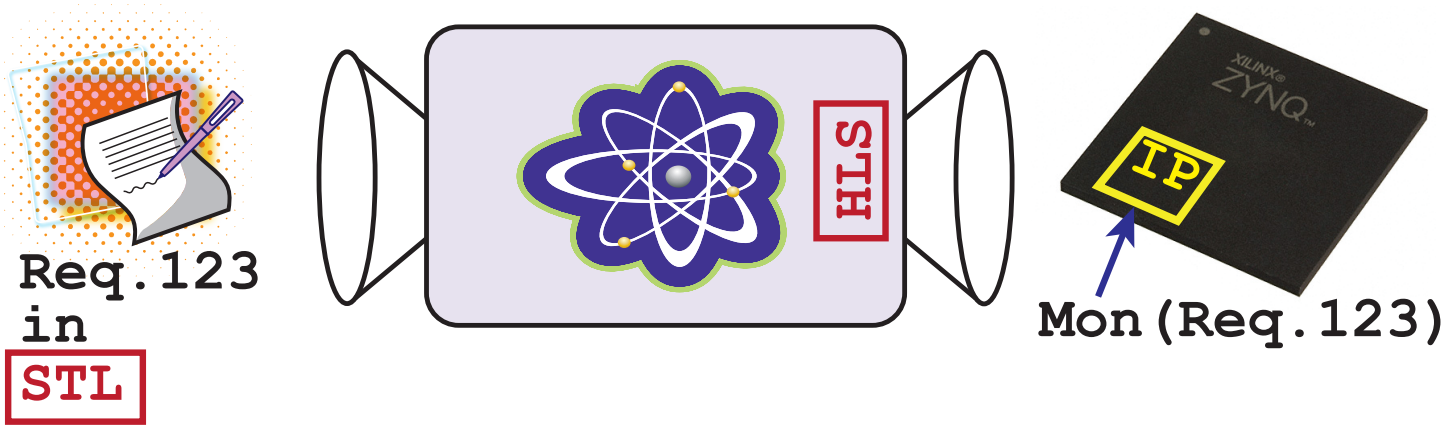
$$\diamondsuit_{\{9\}} \uparrow \ell \rightarrow \diamondsuit_{[0,4]} \left(\diamondsuit_{\{5\}} \uparrow f \wedge \boxminus_{[0;5]} \neg d \right)$$

Recap

The takeaway message:

- From system level requirements to hardware monitors
- Signal Temporal Logic as a specification language
- High Level Synthesis for HDL generation



2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

References I


Martin Leucker.
Teaching Runtime Verification.
In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verification*, volume 7186 of *Lecture Notes in Computer Science*, pages 34–48. Springer Berlin Heidelberg, 2012.