2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Automated Safety Verification for Automotive Microcontrollers

Holger Busch

Infineon Technologies

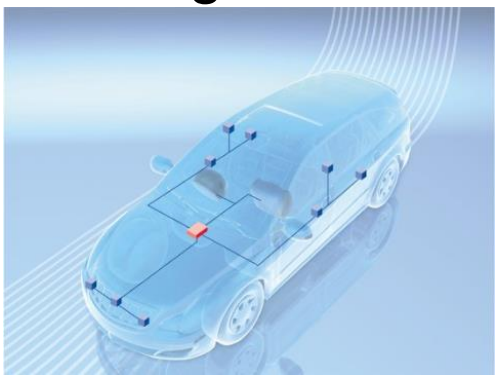
2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Contents

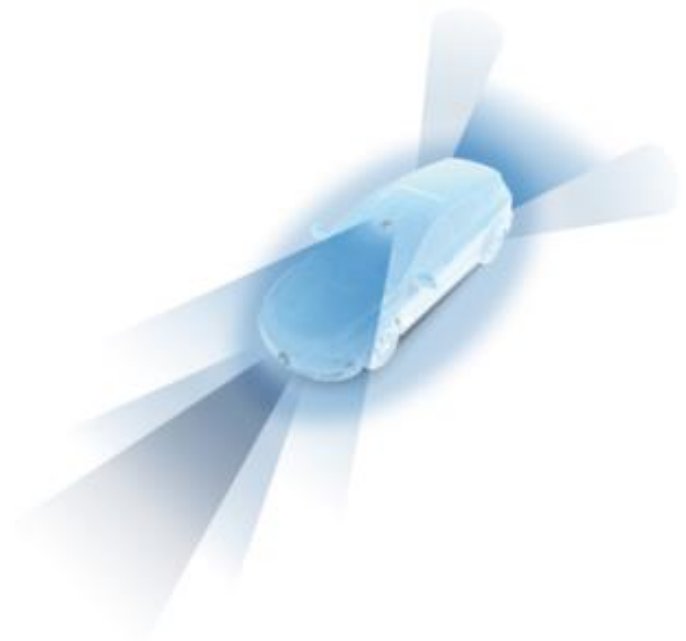
- Motivation
- ISO26262
- Hardware Safety Measures
- Formal Safety Verification
- Formal Fault Injection
- Safety-Modifications
- Results
- Conclusions

Motivation

- Growing number of safety applications for µC:

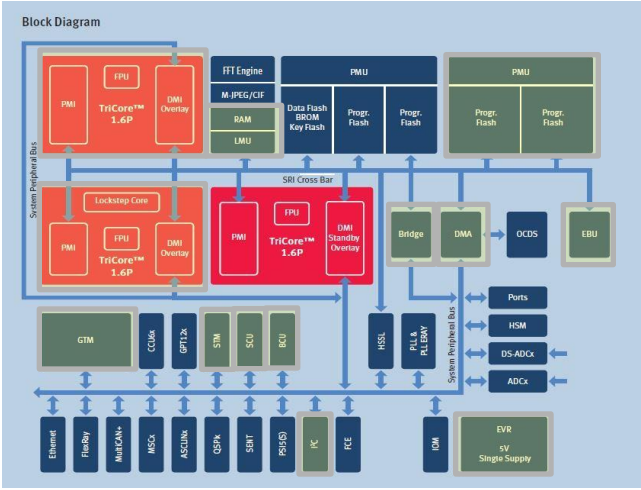


  - Restraint Systems (e.g. Airbag)
  - **E**lectric **P**ower **S**teering
  - Electro Hydraulic Power Steering
  - Chassis Domain Control
  - ABS/VSC
  - Vehicle Stability Control (ESP)
  - Suspension Control
  - **A**dvanced **D**river **A**ssistant **S**ystems



ISO 26262 Standard

- Mass-produced passenger cars (<3.5 t)
- ASIL classification of automotive µC parts
  - Safety concept, with ASIL decomposition
- HW and SW safety measures
- ISO 26262 certification
  - V-model for development process
  - Diagnostic coverage analysis
  - Documentation, requirement traceability
- Safety verification on top of functional verification

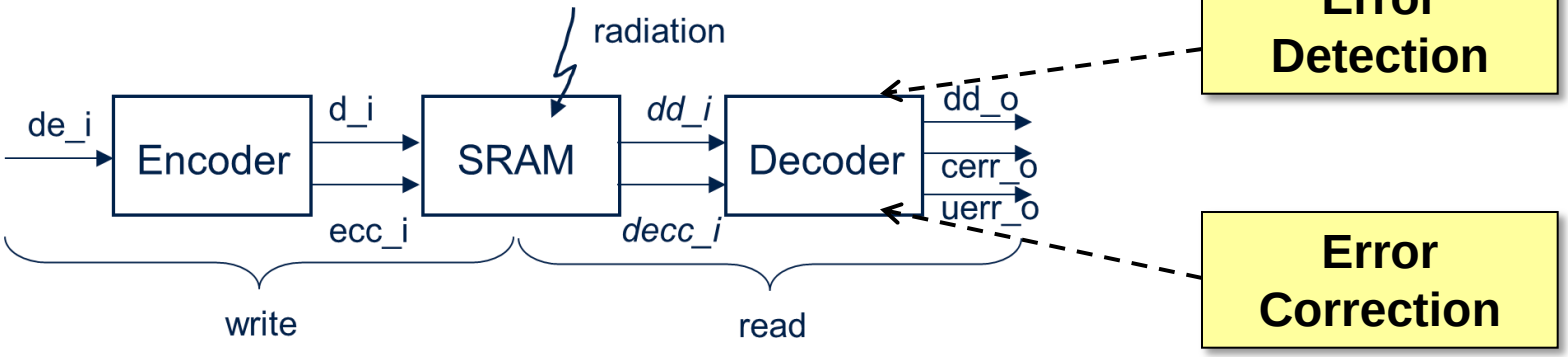


➢ Higher product complexity, area and power consumption, higher development and production costs

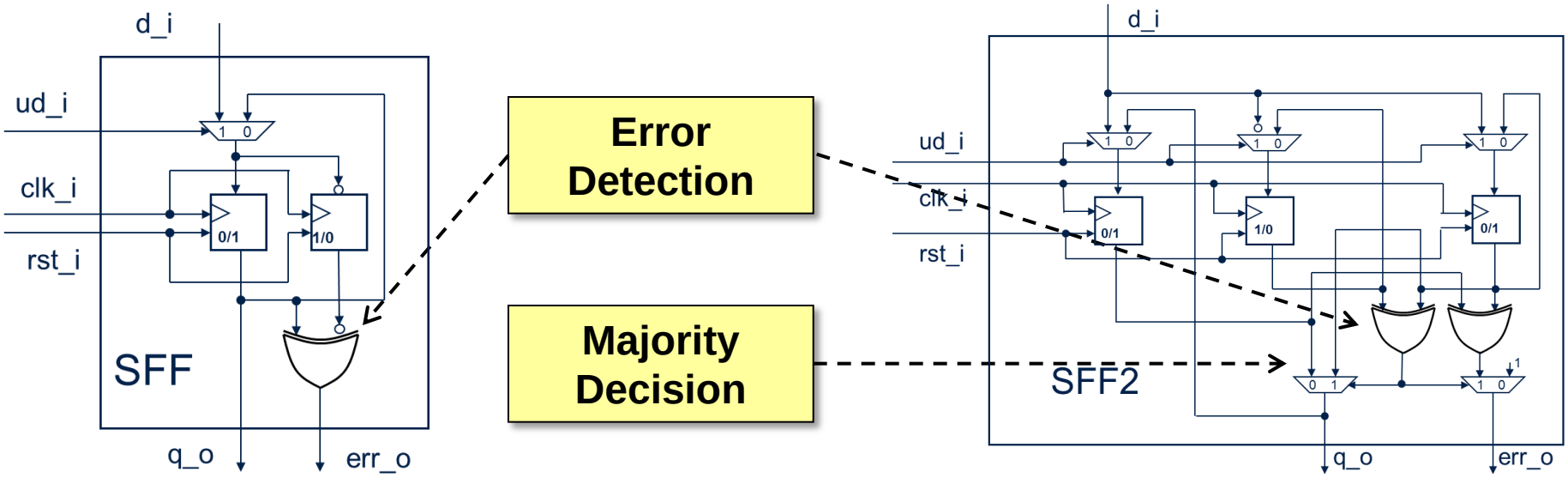
No choice: ISO-compliance mandatory!

HW Safety Measures

- Error detection/correction codes

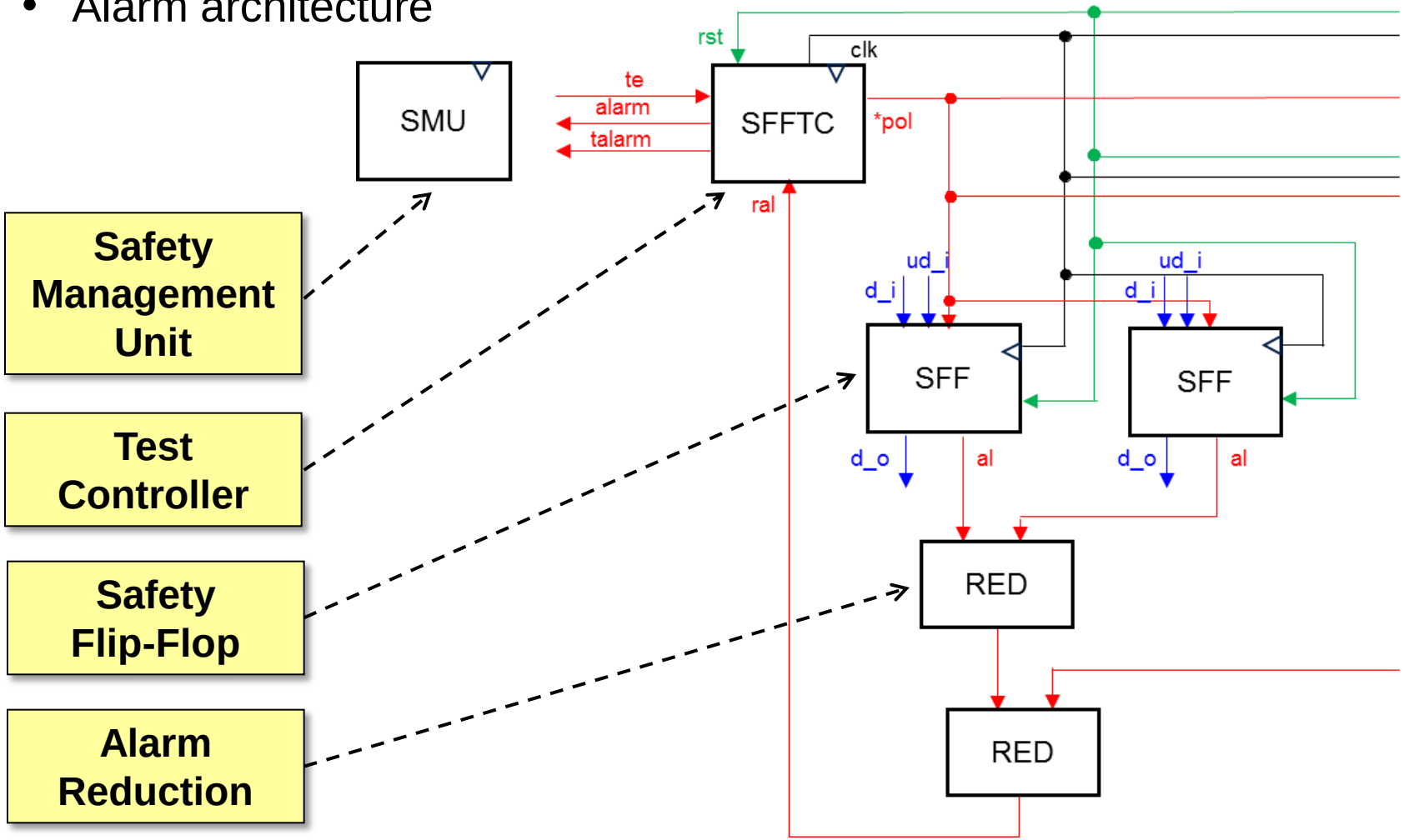


- Double/Triple Redundancy



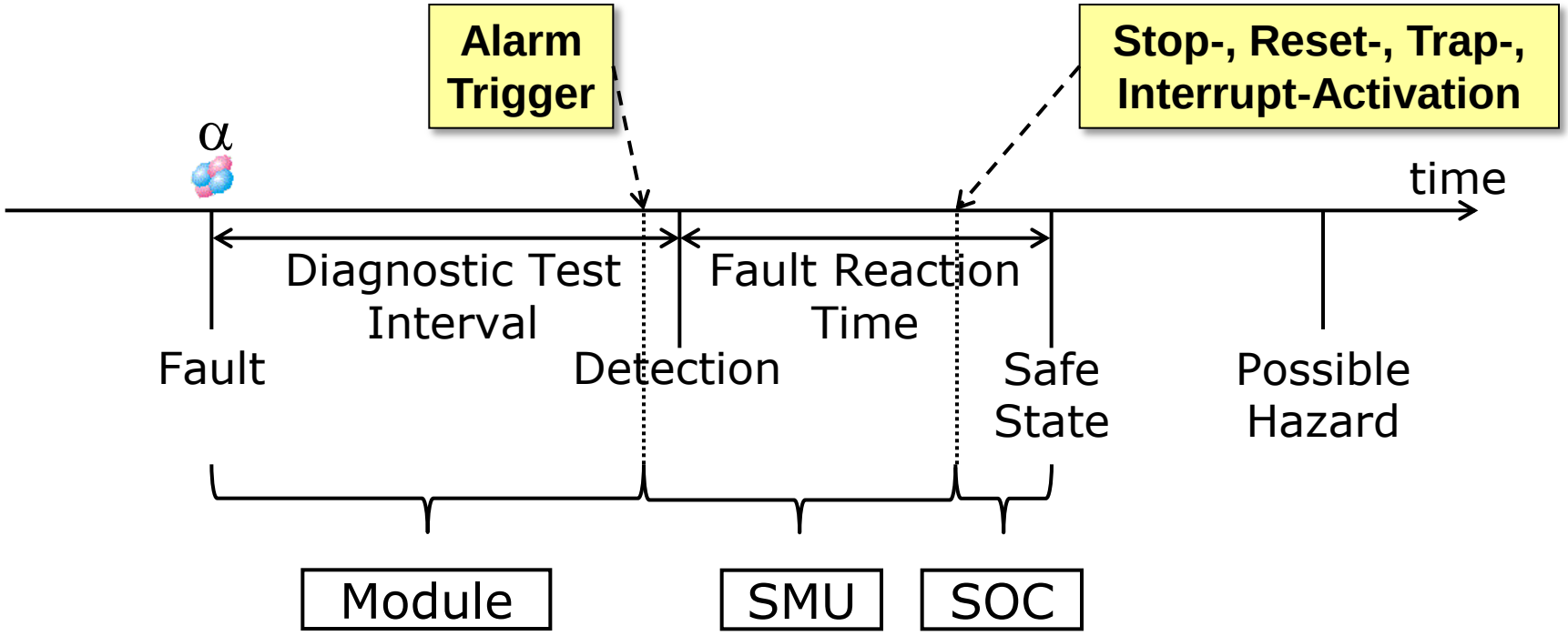
HW Safety Measures (1)

- Alarm architecture



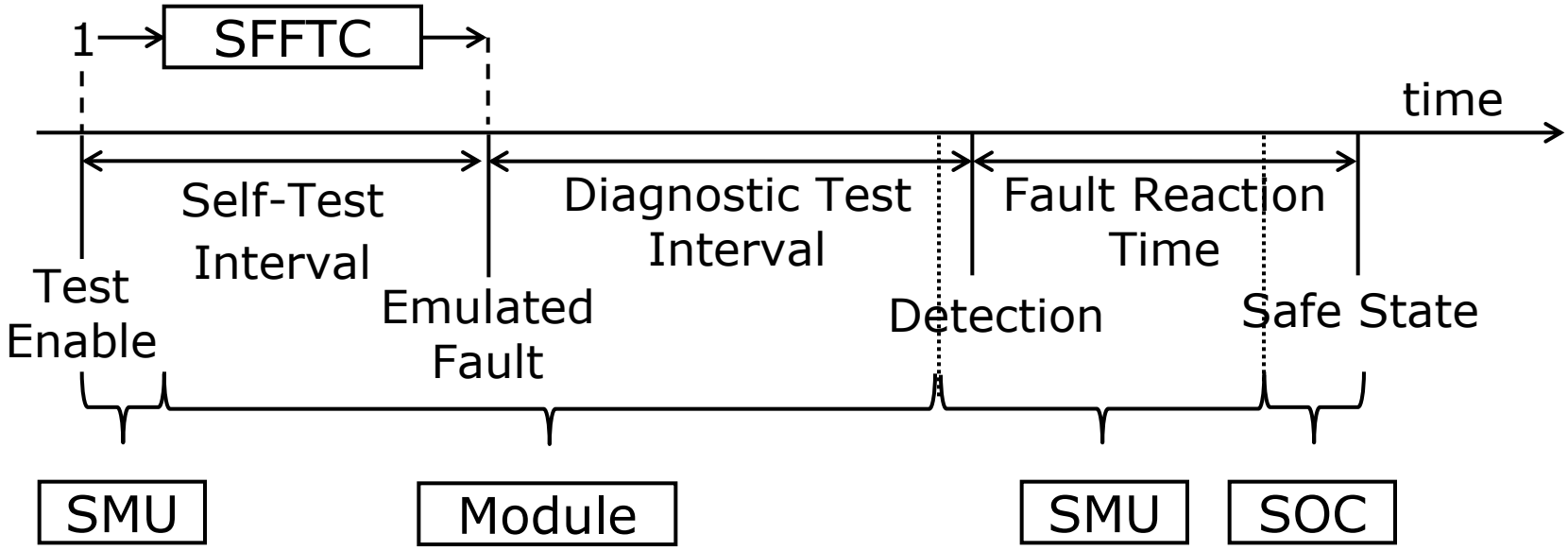
Formal Safety Verification

- Phases after fault occurrence:
  - Diagnostic test interval until detection
  - Fault reaction time until safe state reached



Formal Safety Verification (1)

- Verify module-part of safety mechanism:
  - Self-test feature
  - Alarm generation



Formal Safety Verification (2)

- Basic formal property scheme:

  fault occurrence,

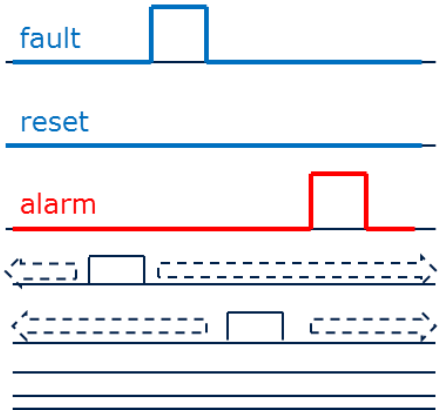
  additional side conditions

  ⇒ expected reaction

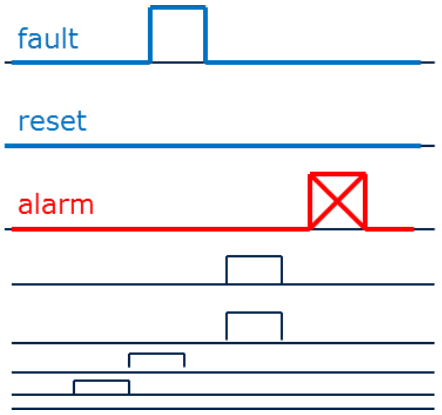
  – Side conditions:
    - No reset; environment constraints,…

  – Expected reaction:
    - Error flag set, alarm pulse sent
    - Protocol violation at module interface

  – Formal property succinct, yet powerful
    - Covers all scenarios
    - If disproven, corner case is shown



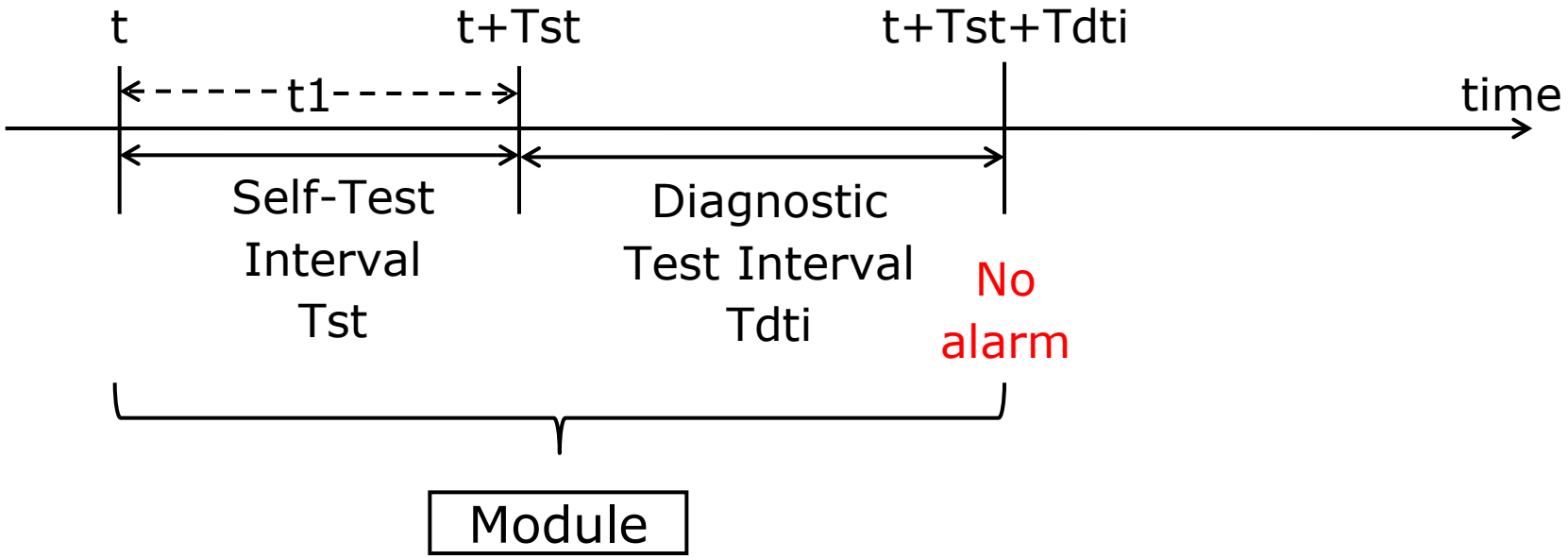
Any Behavior Covered



Witness Trace

Formal Safety Verification (3)

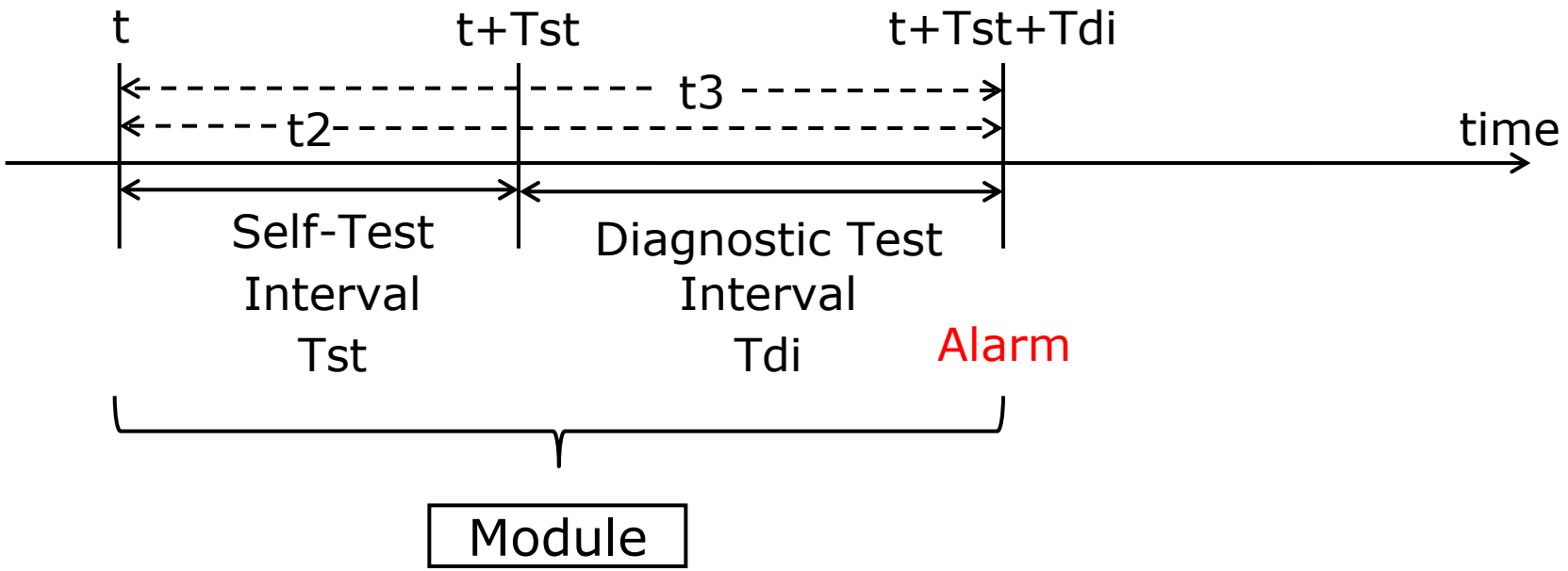
no test ⇒ no (false) alarm (1)



$\forall_{t1:\ t \le t1 \le t+Tst}$ te(t1) = 0 ⇒ alarm(t+Tst+Tdti) = 0 (1)

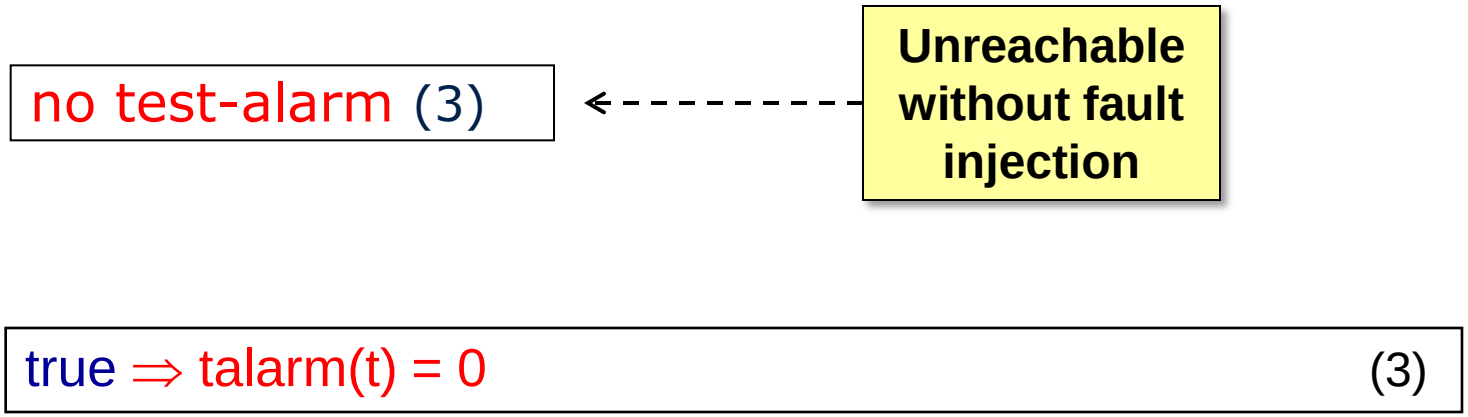
Formal Safety Verification (4)

test enabled ⇒ alarm (2)



$te(t)=1 \wedge \forall_{t2:\, t \le t2 \le t+Tst+Tdi}\ reset(t2) = 0 \Rightarrow \exists_{t3:\, t \le t3 \le t+Tst+Tdi}\ alarm(t3) = 1$ (2)

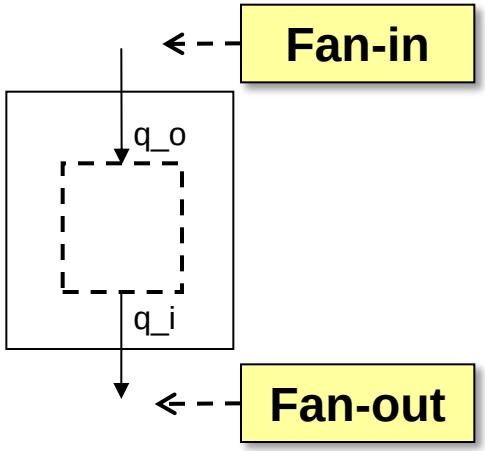
Formal Safety Verification (5)



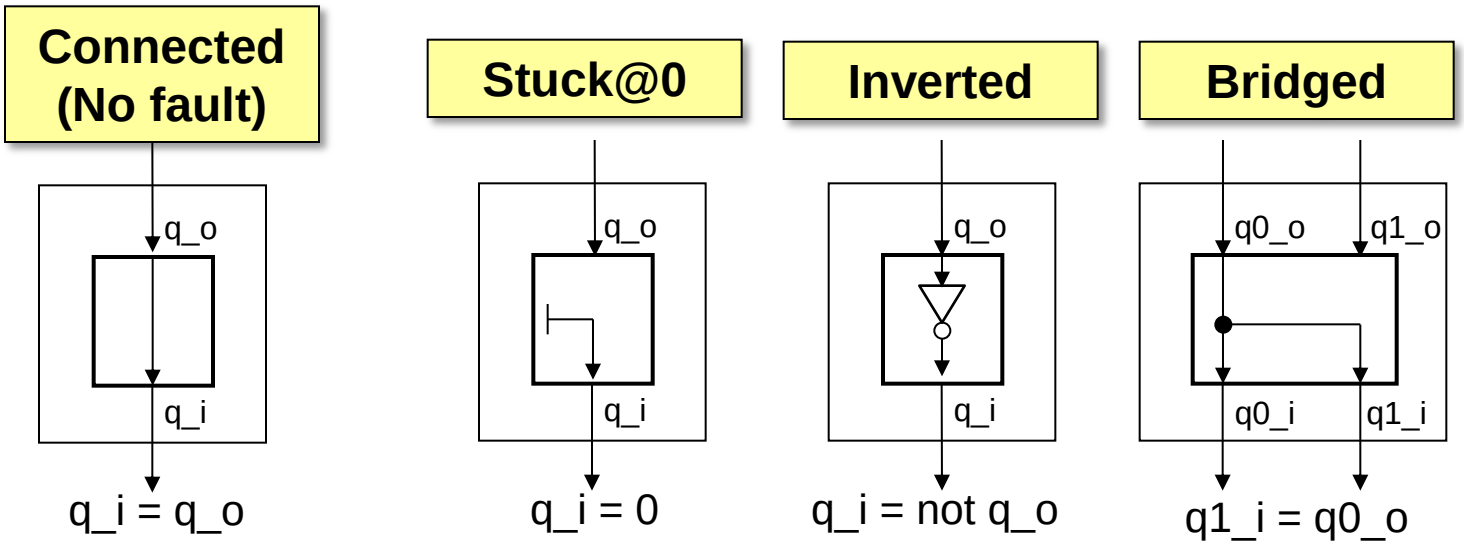
true ⇒ talarm(t) = 0 (3)

Formal Fault Injection

- Onespin‘s cut-option for compilation :
  - List of signals to be cut in model
  - Fan-in-part: real behavior according to design (output)
  - Fan-out-pat: behavior arbitrarily constrainable (input)

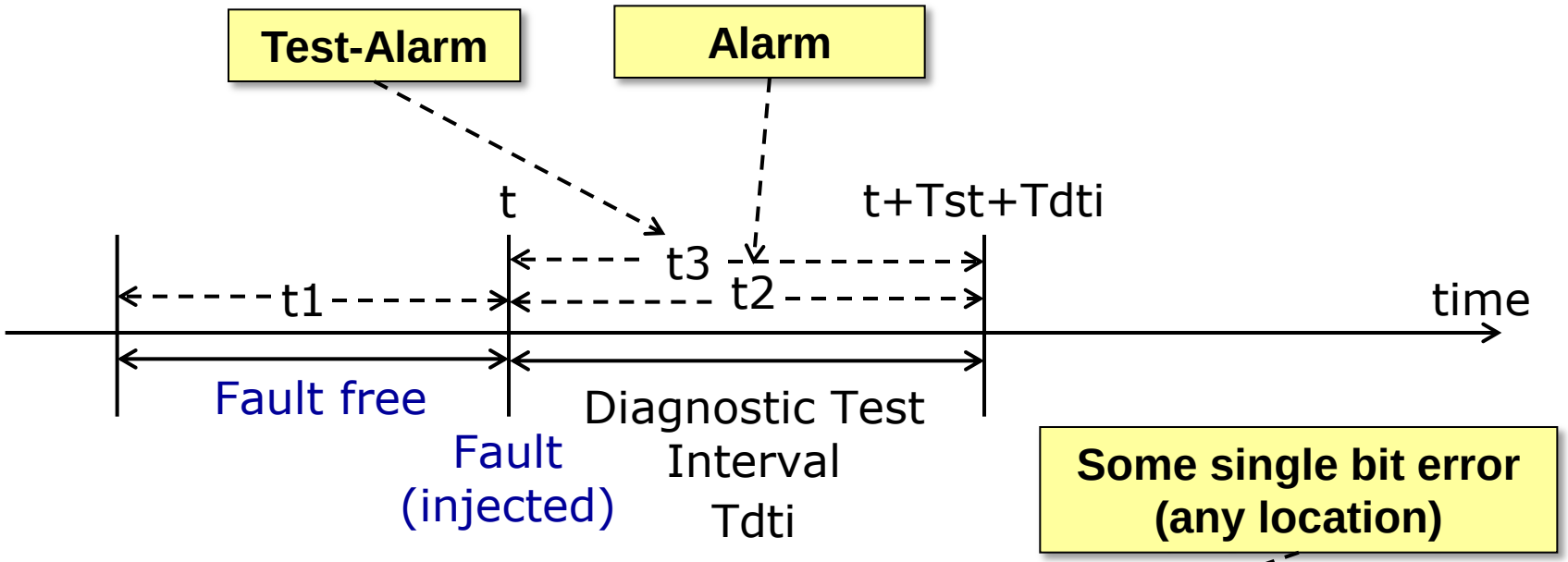


- Fault models set by formal assumption in property



Formal Fault Injection (1)

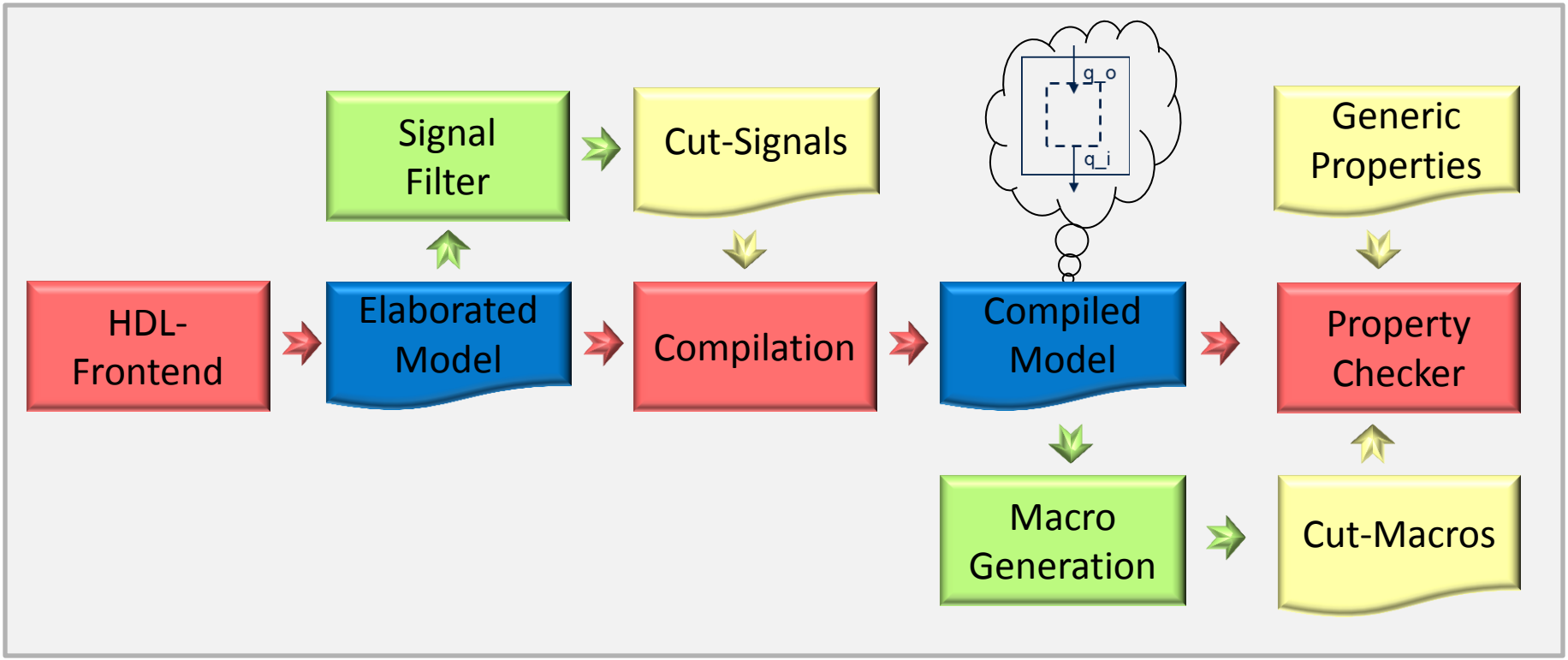
fault ⇒ alarm and test-alarm (4)



$$\forall_{t1:\, t1<t}\ be(0)(t1) \wedge talarm(t1) = 0 \wedge be(1)(t)$$
$$\Rightarrow \exists_{t2:\, t\le t2\le t+Tdti}\ alarm(t2) = 1 \wedge \exists_{t3:\, t\le t3\le t+Tdti}\ talarm(t3) = 1 \qquad (4)$$

Formal Fault Injection: Flow

- Automatic signal filtering for injection
- Automatic macro generation
- Pre-defined generic properties

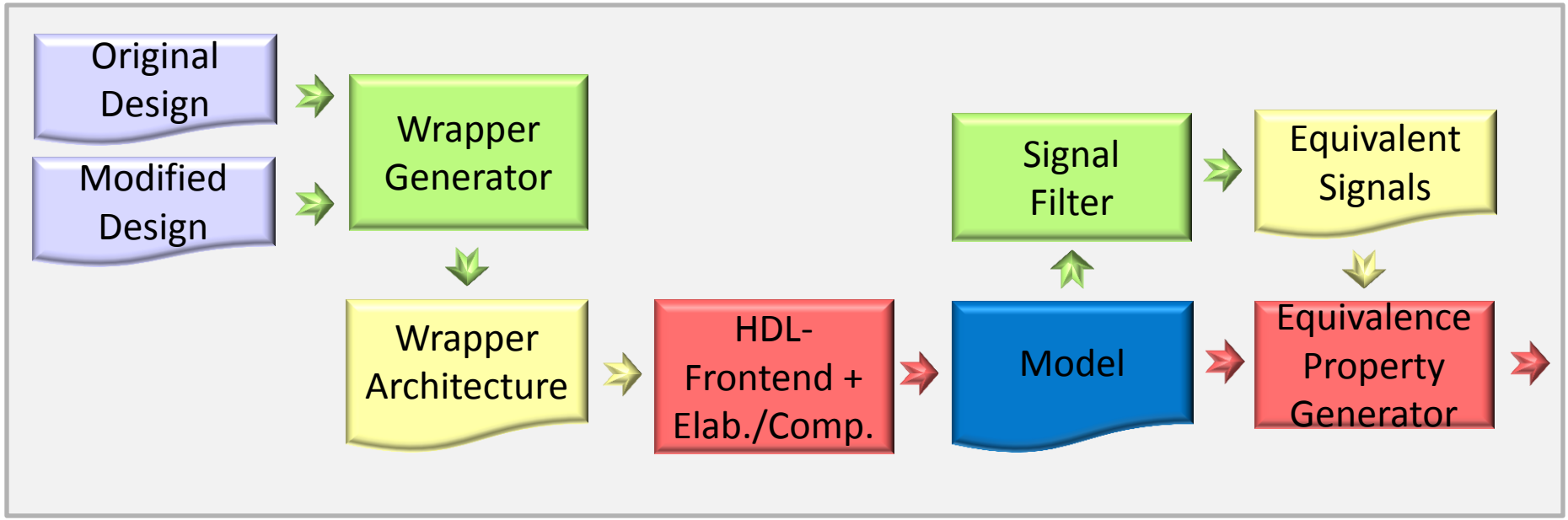


Safety Modifications

- Due to changed safety requirements and analyses
  - New safety application requires inclusion of more registers
  - New SW-mechanism allows removal of HW-safeguarding
  - Product derivative for customer with non-safety application
  - Safety measure found to be missing, or superfluous
- Need to ensure:
  - Mission function not affected
- Verification approaches:
  - Rerun regression with complete coverage (-> Certitude, Quantify)
  - Equivalence check focussed on input-output behaviour of modified sub-components

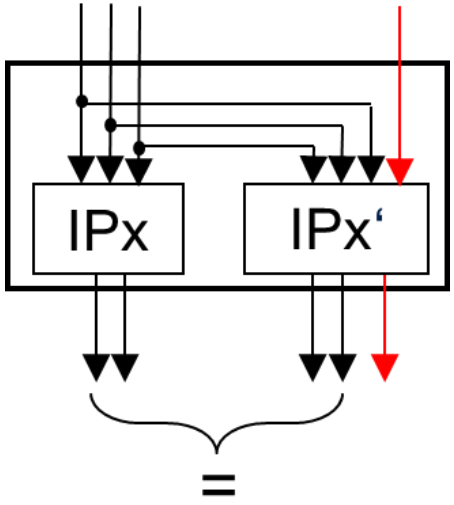
Safety Modifications (1)

- Equivalence checking flow:
  - Generation of wrapper with both design versions
  - Generation of equivalence properties



Safety Modifications (2)

- Property-based equivalence checking:



All corresponding outputs

$ipx\_o(t) = ipx'\_o(t)$ (8)

Base-case

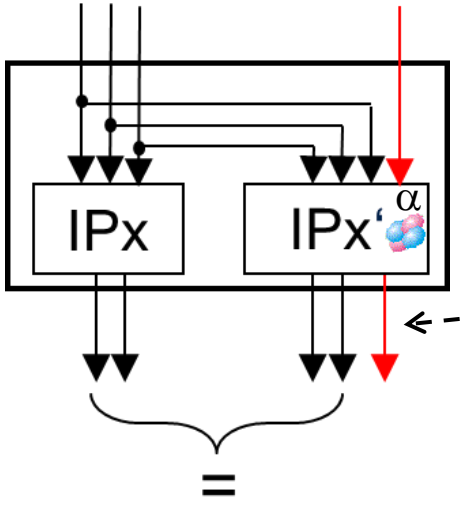
Step-case

$reset(t) = 1 \Rightarrow ipx\_o(t) = ipx'\_o(t) \wedge q(t) = q'(t)$ (8a)

$ipx\_o(t) = ipx'\_o(t) \wedge q(t) = q'(t) \Rightarrow ipx\_o(t+1) = ipx'\_o(t+1) \wedge q(t+1) = q'(t+1)$ (8b)

Safety Modifications (3)

- Injection of correctable errors:



Alarms triggered

Mission function not affected!

$$\forall_{n:\, n \le nc}\ \forall_{t1:\, t1 < t}\ be(0)(t1) \wedge talarm(t1) = 0 \wedge be(n)(t)$$
$$\Rightarrow\ \forall_{t4:\, t \le t4 \le t+Tdti}\ ipx\_o(t4) = ipx'\_o(t4) \wedge q(t4) = q'(t4)$$
$$\exists_{t2:\, t \le t2 \le t+Tdti}\ alarm'(t2) = 1 \wedge \exists_{t3:\, t \le t3 \le t+Tdti}\ talarm'(t3) = 1 \quad (9)$$

Results - 2436 Fault Locations

Property	Status	MemoryPeak(MB)	Proof Time (hh:mm:ss)
sff_0_be_alarm_rst	hold	2479.07	00:01:22
sff_0_be_alarm_step	hold	2250.46	00:02:33
sff_0_be_no_alarm_rst	hold	2552.25	00:13:44
sff_0_be_no_alarm_step	hold	14199.30	02:24:11
sff_0_be_no_talarm_rst	hold	1990.68	00:21:01
sff_0_be_no_talarm_step	hold	2635.84	00:01:57
sff_0_be_ok_rst	hold	1597.71	00:00:04
sff_0_be_ok_step	hold	8280.96	00:00:26
sff_1_be_alarm_rst	hold	1698.65	00:00:19
sff_1_be_alarm_step	hold	7812.13	00:01:45
sff_1_be_talarm_rst	hold	1685.32	00:00:17
sff_1_be_talarm_step	hold	8544.13	00:01:17
sff_any_be_alarm_rst	hold	1682.46	00:00:07
sff_any_be_alarm_step	hold	7598.20	00:04:11
sff_any_be_talarm_rst	hold	1528.82	00:00:44
sff_any_be_talarm_step	hold	7546.81	00:01:27

Results (1)

- Limited set of re-usable properties sufficient
- Astronomical # fault combinations exhaustively checked in reasonable time: 2**2436 in 4 min
- More expensive to prove that no false alarm
- Partitioned fault sets so far not needed
- Typical bugs found:
  - Not all SFF-bits included in alarm reduction
  - Clock domain discrepancies
  - Incorrect SFF-integration

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Conclusions

- Exhaustive formal safety verification extremely fast
  - 100% diagnostic coverage proven
- High proof automation by generic pre-defined properties and design-specific generated macros
- Equivalence checks of safety modifications efficient
  - Focused on modified sub-components
  - Complete debugging support of property checker
  - Applicable if no complete regression suite available
- Automatic safety verification functions tailorable to specific HW safety mechanisms