Safety-Verification Flow Sporting Gate-Level Accuracy and Near Virtual-Prototype Speed

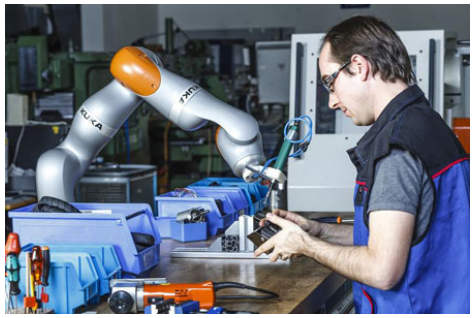
B.-A. Tabacaru, M. Chaari, W. Ecker,

T. Kruse, and C. Novello

SPONSORED BY THE
Federal Ministry of Education and Research

Functional Safety

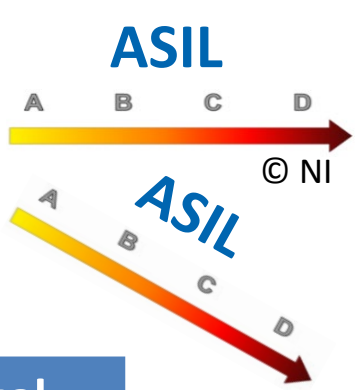
- Comes more and more in focus since automation increasingly impacts our daily lives
- Is already well established in the aerospace industry
- Requests designs to be more than correct
  - **Fail operational**: Nothing bad happens
  - **Fault tolerant**: The device does not fail due to fault occurrence

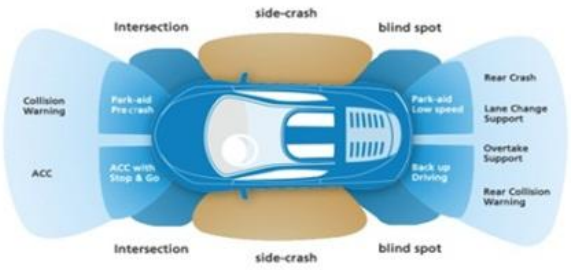


Motivation

ISO 26262

© Safety Lab





© EETimes

› Fault-Injection into

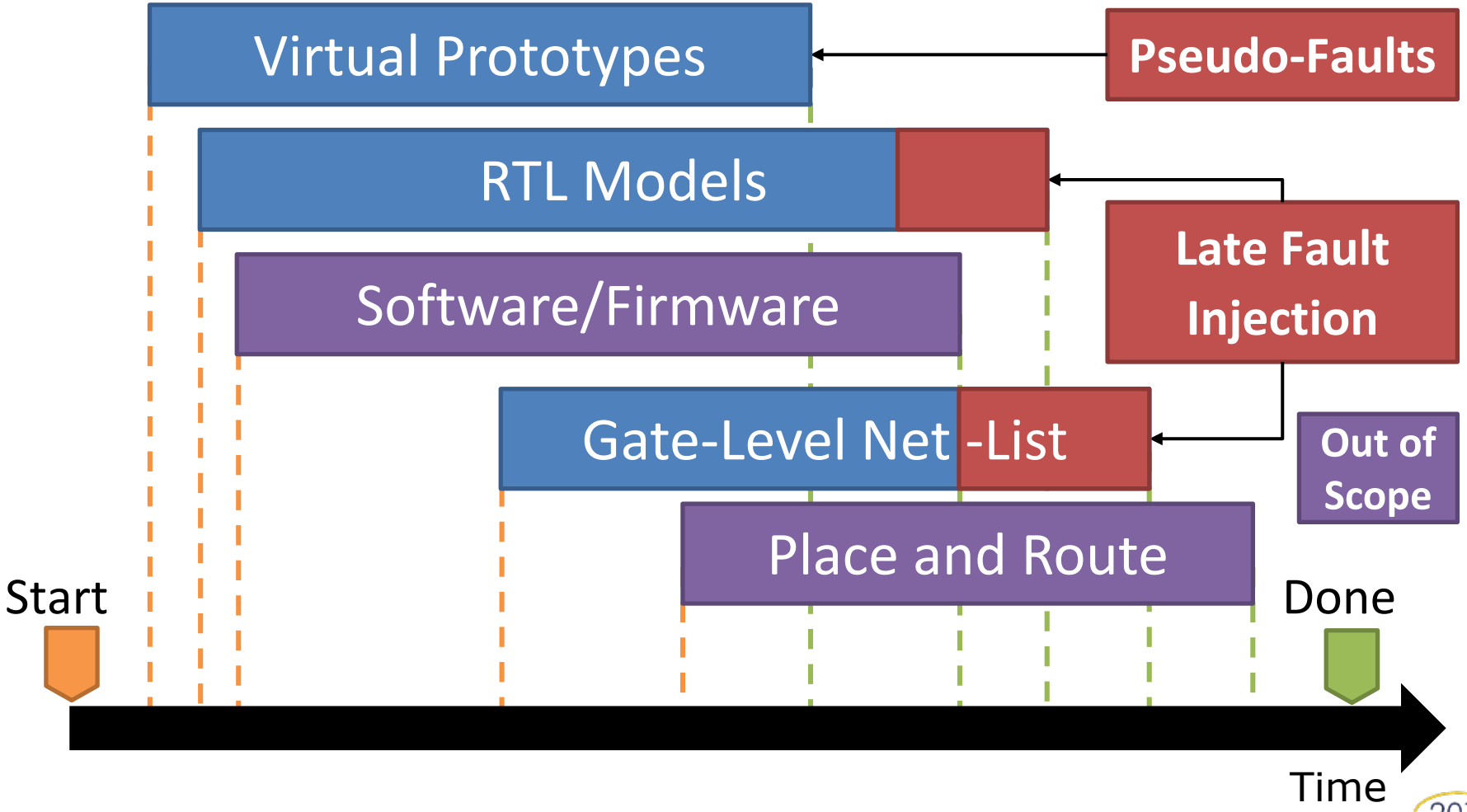
Gate-Level Net-Lists	RTL Models	Virtual Prototypes
✓	✓	✗

- ✗ Late availability
- ✗ Long simulation time
- ✗ High debugging cost
- ✗ Huge number of simulations

- ✓ Early availability
- ✓ Short simulation time
- ✓ Fast debugging
- ✓ Reduced number of simulations

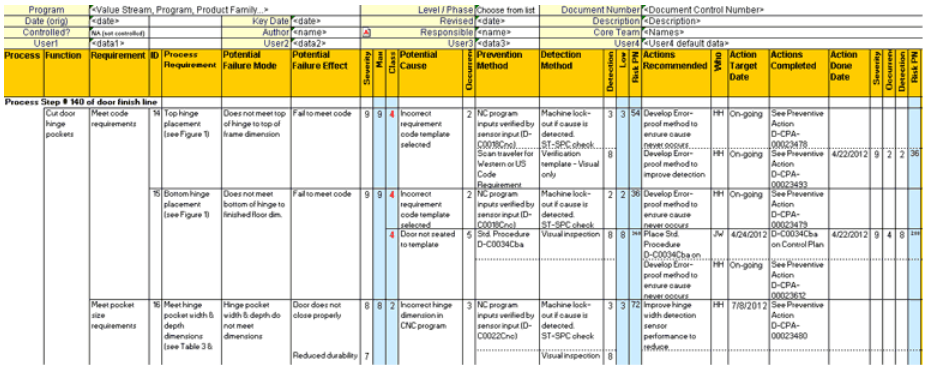
→ ✗ Pseudo-Faults

Standard Development Flow (Simplified Example)



Pseudo-Faults (Trivial Example)

› FMEDA for CPU



Program	<Value Stream, Program, Product Family...>			Level / Phase	Choose from list	Document Number	<Document Control Number>
Date (orig)	<date>		Key Date	<date>	Revised	<date>	Description	<Description>
Controlled?	NA (not controlled)		Author	<name>	Responsible	<name>	Core Team	<Names>
User1	<data1>		User2	<data2>	User3	<data3>	User4	<User4 default data>

Process	Function	Requirement	ID	Process Requirement	Potential Failure Mode	Potential Failure Effect	Severity	Mode Class	Potential Cause	Occurrence	Prevention Method	Detection Method	Detection	Low Risk	Risk	Actions Recommended	Who	Action Target Date	Actions Completed	Action Done Date	Severity	Occurrence	Detection	Risk Pty
Process Step # 140 of door finish line																								
	Cut door hinge pockets	Meet code requirements	14	Top hinge placement (see Figure 1)	Does not meet top of hinge to top of frame dimension	Fail to meet code	9	9	4 Incorrect requirement code template selected	2	NC program inputs verified by sensor input (D-C0018Cnc)	Machine lock-out if cause is detected. ST-SPC check	3	3	54	Develop Error-proof method to ensure cause never occurs	HH	On-going	See Preventive Action D-CPA-00023478					
												Verification template - Visual only	8			Scan traveler for Western or US Code Requirement	HH	On-going	See Preventive Action D-CPA-00023493	4/22/2012	9	2	2	36
			15	Bottom hinge placement (see Figure 1)	Does not meet bottom of hinge to finished floor dim.	Fail to meet code	9	9	4 Incorrect requirement code template selected	2	NC program inputs verified by sensor input (D-C0018Cnc)	Machine lock-out if cause is detected. ST-SPC check	2	2	36	Develop Error-proof method to ensure cause never occurs	HH	On-going	See Preventive Action D-CPA-00023479					
									4 Door not seated to template	5	Std. Procedure D-C0034Cba	Visual inspection	8	8	288	Place Std. Procedure D-C0034Cba on Control Plan	JW	4/24/2012	D-C0034Cba on Control Plan	4/22/2012	9	4	8	288
																Develop Error-proof method to ensure cause never occurs	HH	On-going	See Preventive Action D-CPA-00023612					
		Meet pocket size requirements	16	Meet hinge pocket width & depth dimensions (see Table 3 b.	Hinge pocket width & depth do not meet dimensions	Door does not close properly	8	8	2 Incorrect hinge dimension in CNC program	3	NC program inputs verified by sensor input (D-C0022Cnc)	Machine lock-out if cause is detected. ST-SPC check	3	3	72	Improve hinge width detection sensor performance to reduce	HH	7/8/2012	See Preventive Action D-CPA-00023480					
						Reduced durability	7					Visual inspection	8											

Source: http://www.systems2win.com/images/deliverables/FMEA_766.gif

› Assumption

Failure Mode: Wrong Program Counter

› Mitigation

Apply ECC on all program-counter bits

› Fault Injection into CPU

› Results

Program Counter is incremented by 4. Thus, bits 0 and 1 never fail

› Bottom Line

Faults injected into bits 0 and 1 are pseudo-faults

Goals and Contributions

Goals

- Improve the confidence level of safety verification on the virtual-prototype abstraction level
- Avoid insertion of pseudo-faults into virtual prototypes

Contributions

- Fault-injection methodology on virtual prototypes and gate-level net-lists
- Cross-reference of fault-injection results across multiple abstraction levels
- Validate and invalidate faults on the virtual-prototype level
- Add fault-injection locations to existing SystemC/TLM-based prototypes

Outline

- Safety-Verification Methods
  - Based on execution
  - Based on fault-injection mechanism
- Mixed-Level Co-Simulation
  - Safety verification on transaction-level (TL) models
  - Enhancing TL models with gate-level net-list information
  - Fault injection into gate-level-accurate TL models
- Safety-Verification Platform
- Research Questions
- Experimental Setup
  - Adder modules
  - In-house MIPS processor
- Results and Discussion
- Conclusion

Safety-Verification Methods Based on Execution

› Formal Methods

- ✓ Improved performance and user friendliness
- ✗ Does not scale on large systems

Gate-Level Net-Lists	RTL Models	Virtual Prototypes
✓	✓	✓

› FPGA/GPU-Based Emulation

- ✓ Real-time speed
- ✓ High accuracy
- ✗ Late availability
- ✗ High costs

Gate-Level Net-Lists	RTL Models	Virtual Prototypes
✓	✓	✗

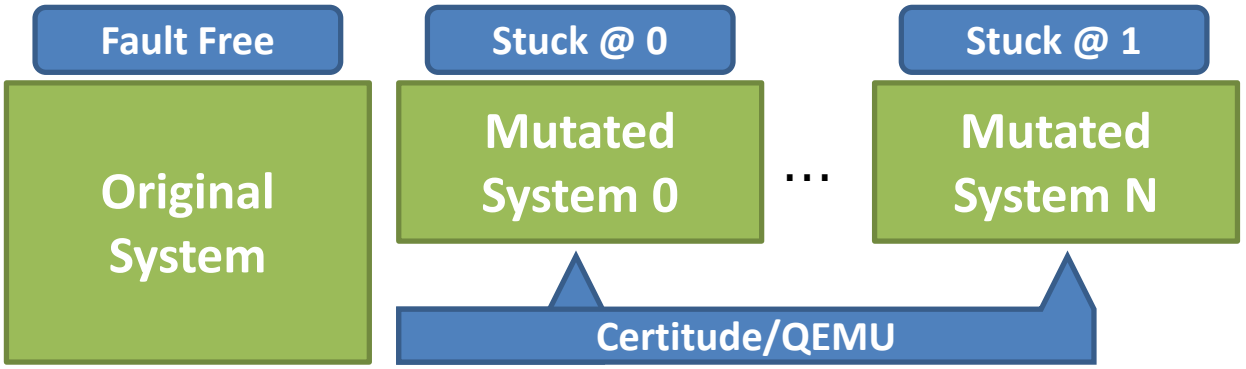
› Simulation Methods

- ✓ High observability and controllability
- ✓ Lower development costs
- ✓ Variable execution speed (based on abstraction level)

Gate-Level Net-Lists	RTL Models	Virtual Prototypes
✓	✓	✓

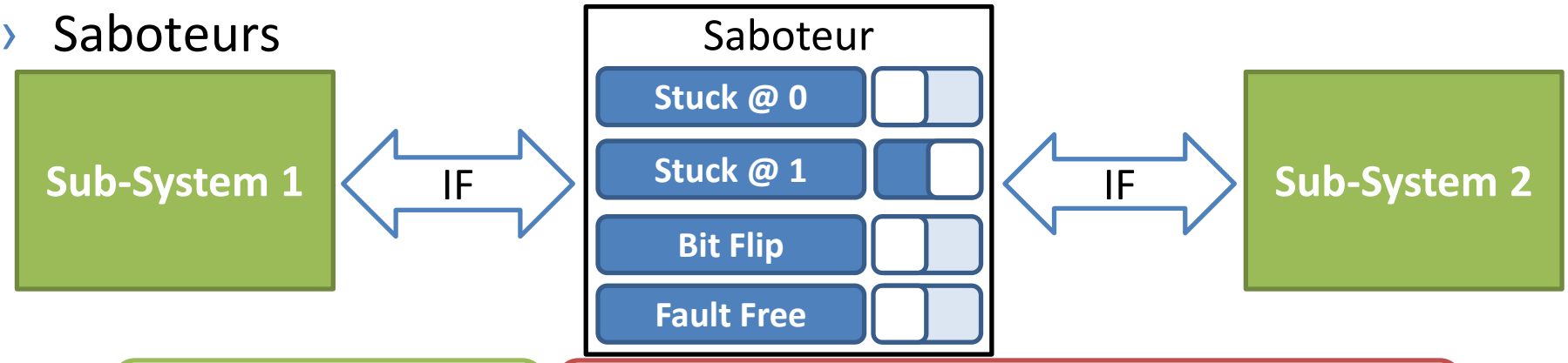
Safety-Verification Methods Based on Fault-Injection Mechanism (I)

- Mutants



- ✖ Only permanent faults supported
- ✖ Modifies the original system structure
- ✖ Pseudo-Faults

- Saboteurs

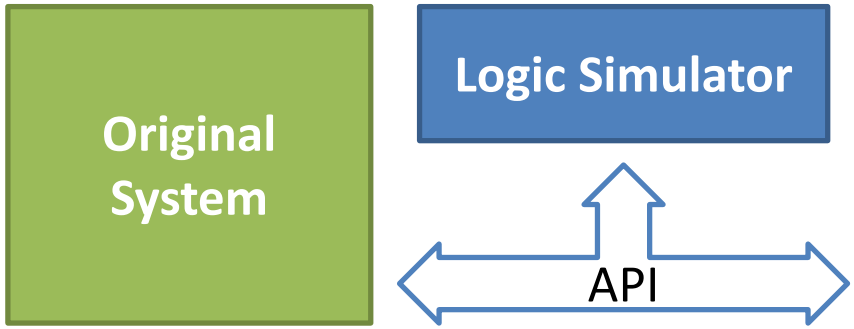


- ✓ Transient faults supported as well

- ✖ Increased net-list area and simulation time
- ✖ Modifies the original system structure

Safety-Verification Methods Based on Fault-Injection Mechanism (II)

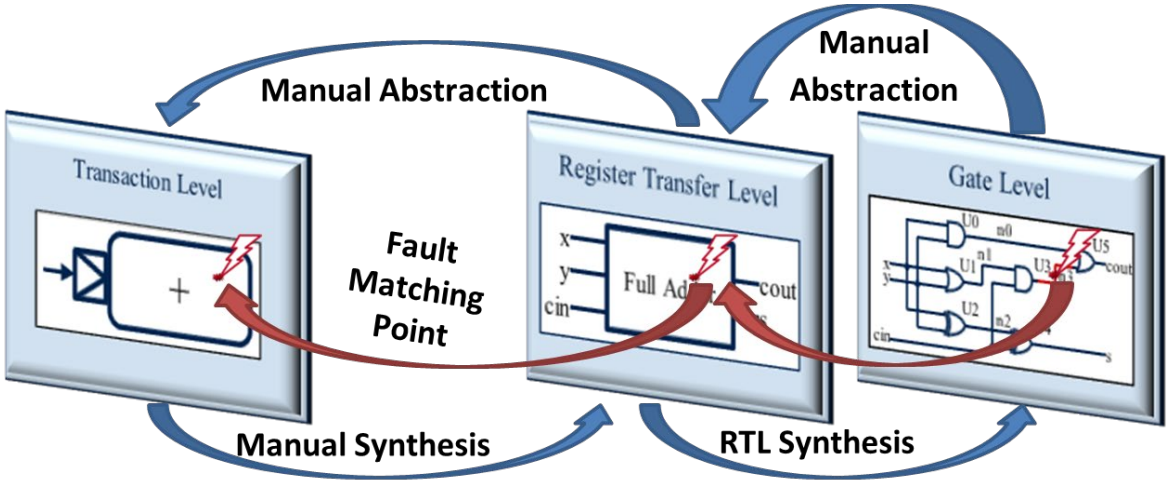
- Simulation Commands



✓ No system changes
✓ Optimized simulation

× Simulator dependent
× Limited support for virtual prototypes

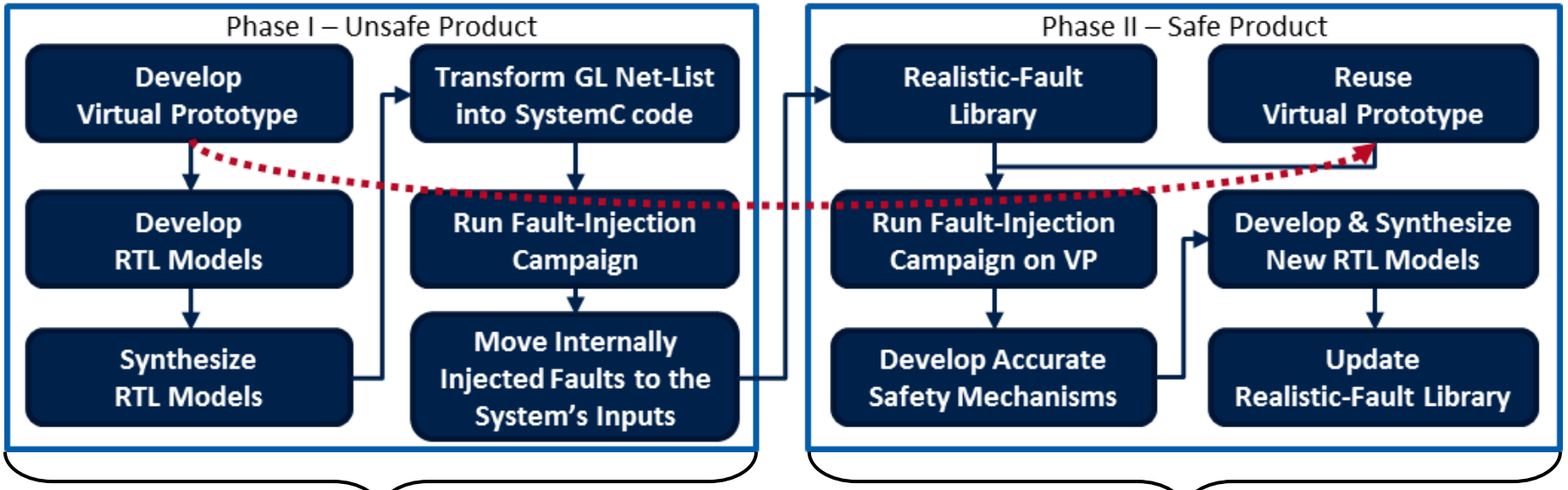
- Mixed-Level Safety Verification



✓ Avoidance of Pseudo-Faults

× Adapters across different models
× Manual implementation

Mixed-Level Co-Simulation



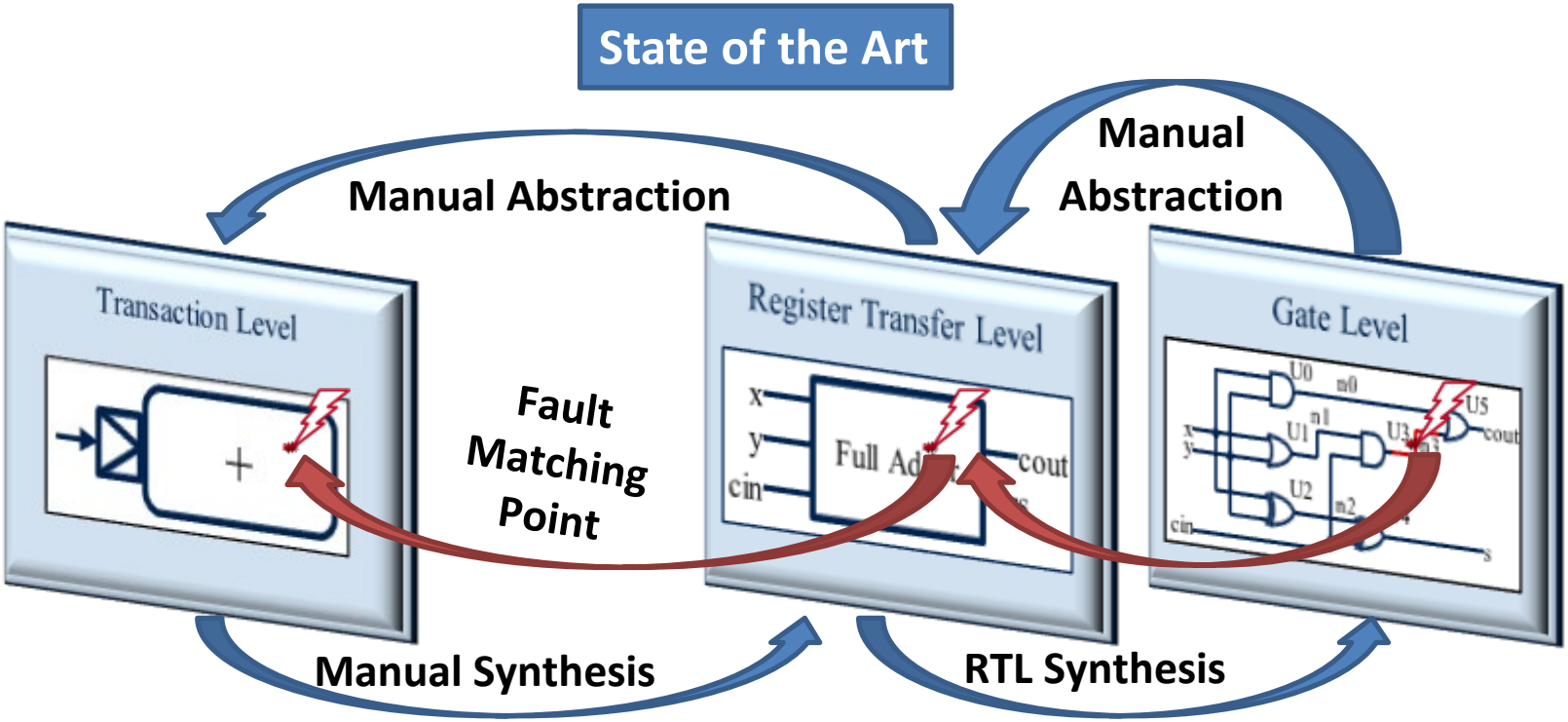
Find fault-sensitive system locations

Make the system robust in an efficient way

✓ Avoid pseudo-faults (i.e., unrealistic faults)

✓ Reusable fault library of realistic faults

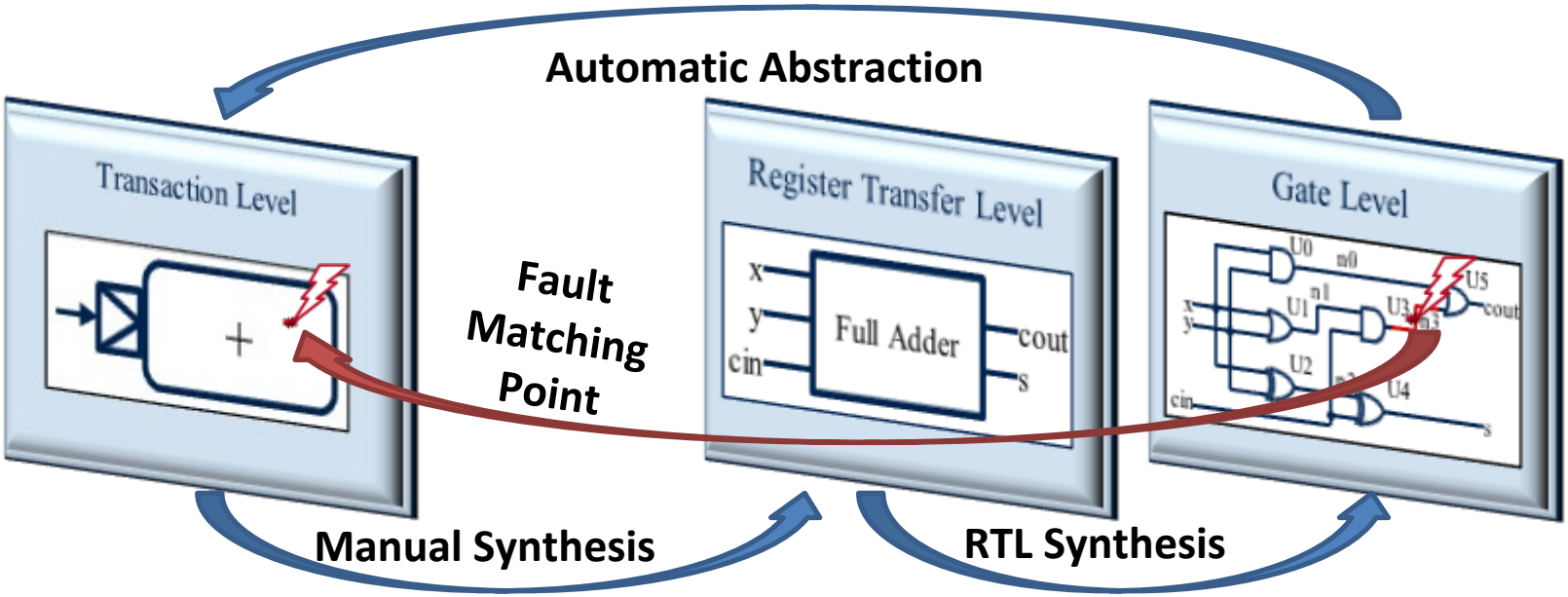
Safety Verification on TLM (I)



- Manual abstraction
- Partial RTL or gate-level implementation details
- No equivalence checking amongst the abstraction levels
- Fault-injection simulations must be run on all abstraction levels

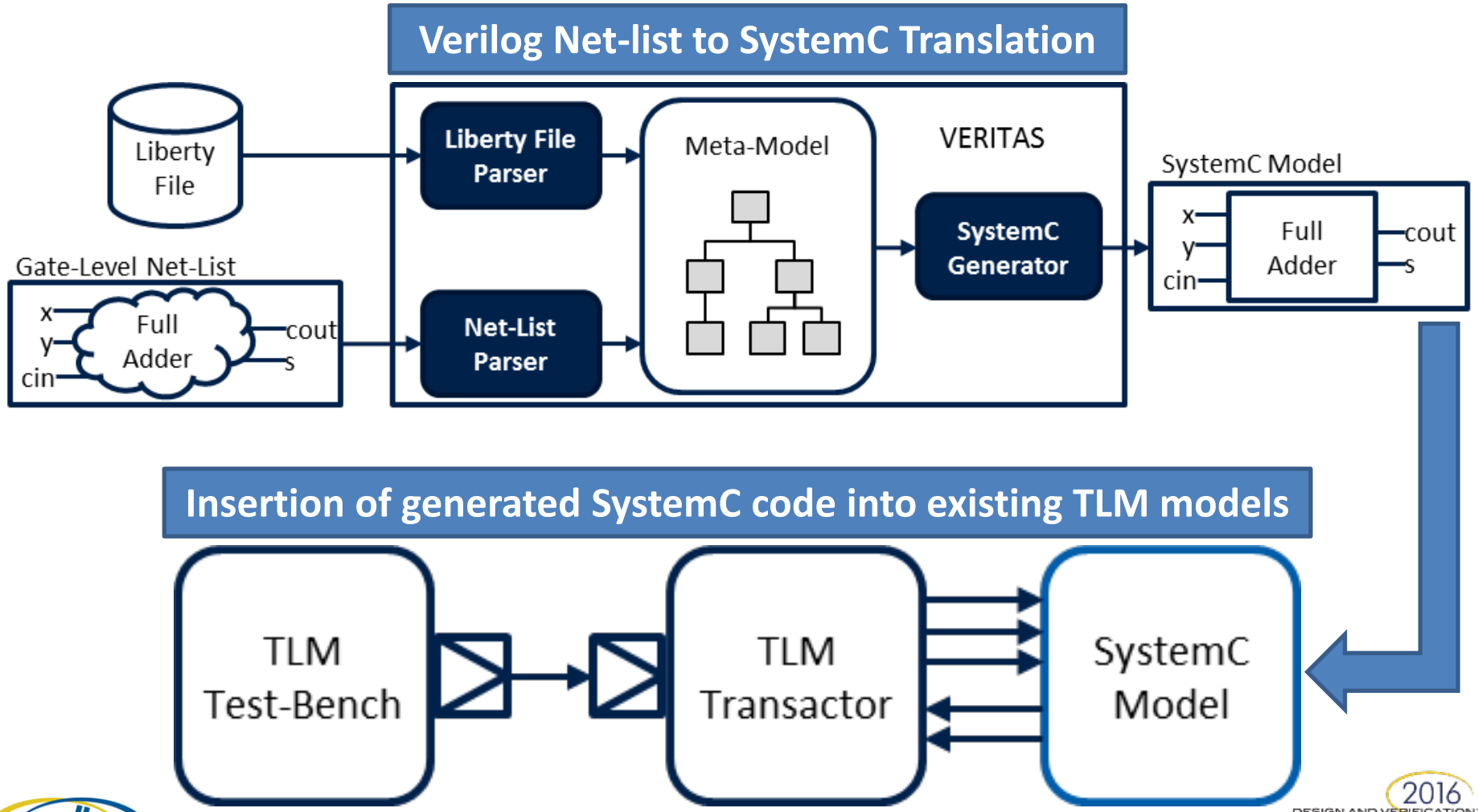
Safety Verification on TLM (II)

Our Approach

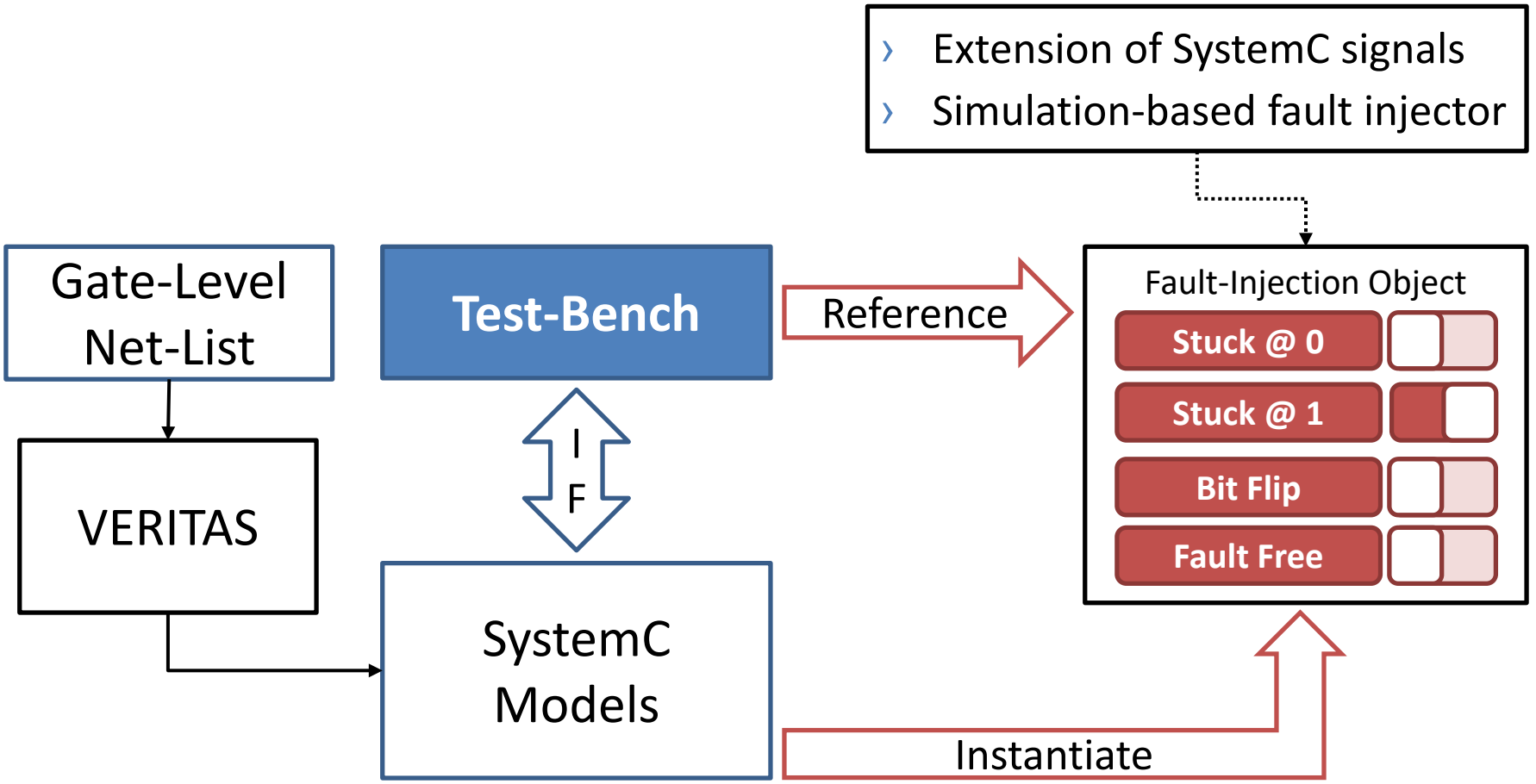


- ✓ **Automatic abstraction**
- ✓ **Gate-level-accurate TL models with all gate-level matching points**
- ✓ **Implicit equivalence checking**
- ✓ **Fault-injection simulation run only on TL abstraction level**

Enhancing TL Models with Gate-Level Net-List Information

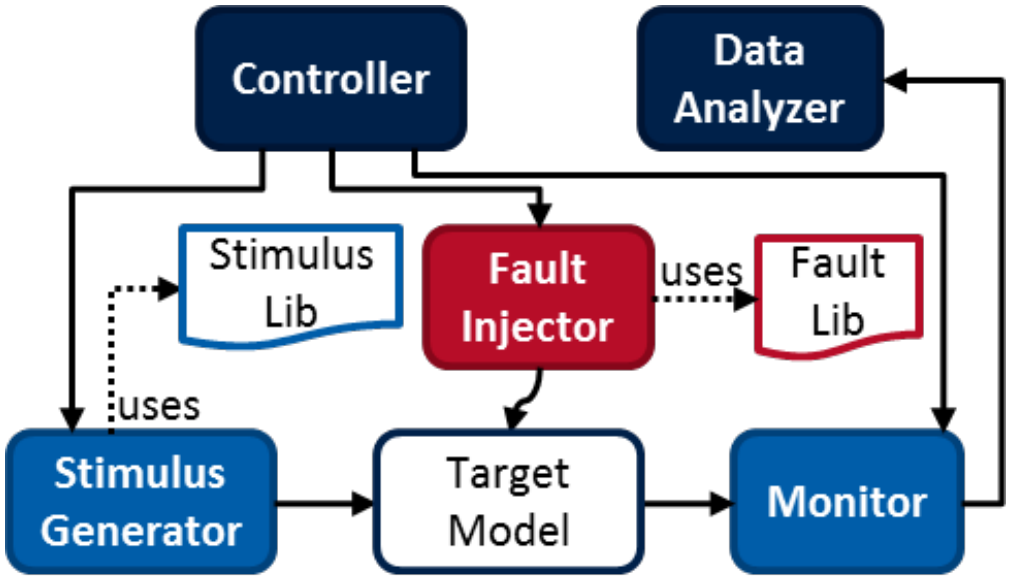


Fault-Injection into Gate-Level Net-Lists

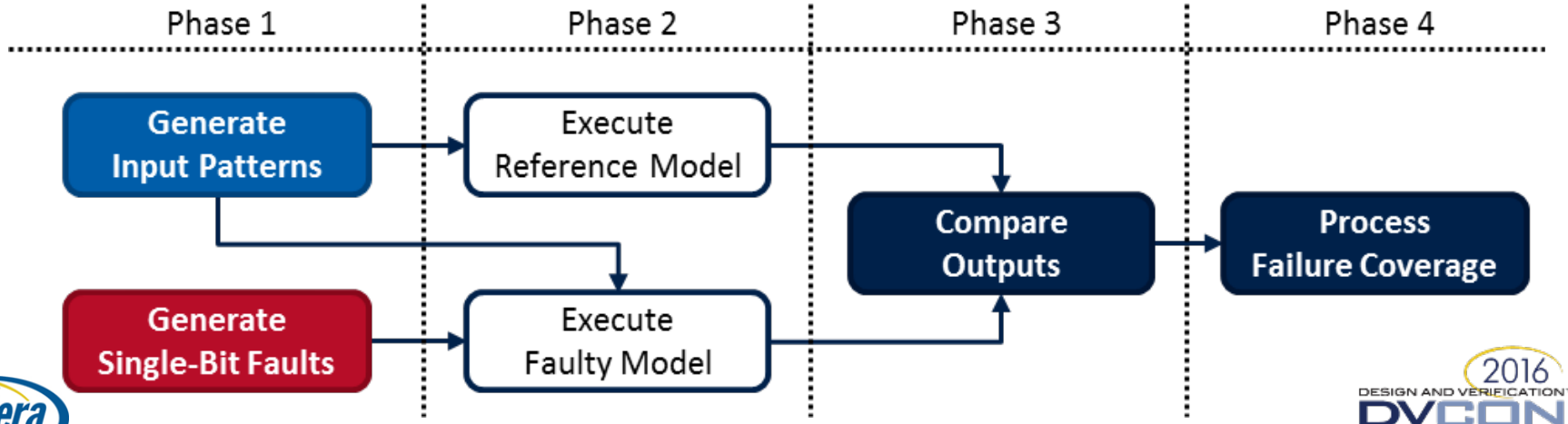


Safety-Verification Platform

Verification Environment



Fault-Injection Flow



Research Questions

- Can the injection of pseudo-faults be avoided on the TLM abstraction level?
- How much can TL models be enhanced with gate-level information?
- What is our approach's impact on simulation performance?

Experimental Setup

- Models
  - 8, 16, 32 bit adders
  - Combinational blocks of in-house MIPS processor
- Monte-Carlo-based fault injection into
  - Internal gate-level signals
  - Inputs of gate-level models
  - Inputs of transaction-level models
  - 10 K, 100 K, 1000 K samples (K = 1000)
- Permanent fault models
  - Stuck-at-0 and stuck-at-1

Results and Discussion

- Injected 10 K, 100 K, and 1000 K faults into combinational gates (adders and MIPS processor)
- 90% of injected faults led to failures
- 2% simulation overhead from fault-injection objects
- **No Pseudo-Faults**

Fault-Injection Locations

Circuits	Correlation Factor		Speed-Up to Gate-Level	Slow-Down to Virtual Protoypes
	Before	After		
Adders	16.48 %	100 %	4.52x	5.91x
MIPS	16.89 %	100 %	66.53x	8.60x

Conclusion

- Safety-verification methodology for virtual prototypes
- Applicable in the early-development cycle of safety-critical systems
- Avoidance of pseudo-faults on virtual prototypes

Questions

Thank you for your attention!