Enhancing Robustness of Sequential Circuits Using Application-specific Know-ledge and Formal Methods

Sebastian Huhn

Stefan Frehse

Robert Wille

Rolf Drechsler

Universität Bremen

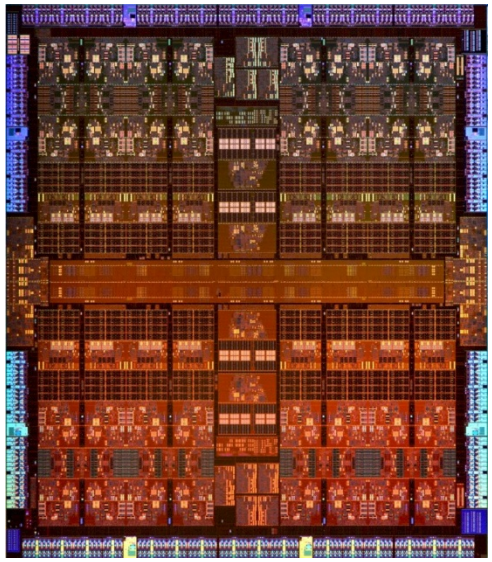
JOHANNES KEPLER UNIVERSITÄT LINZ

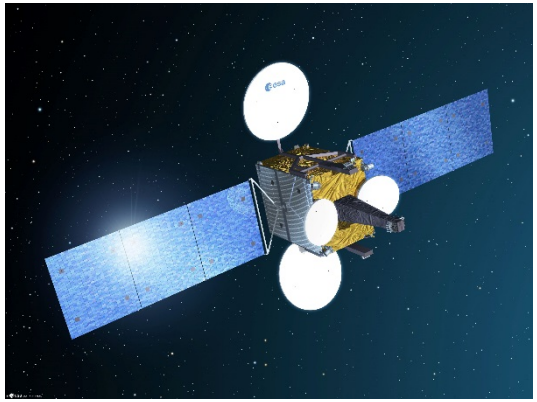
Outline

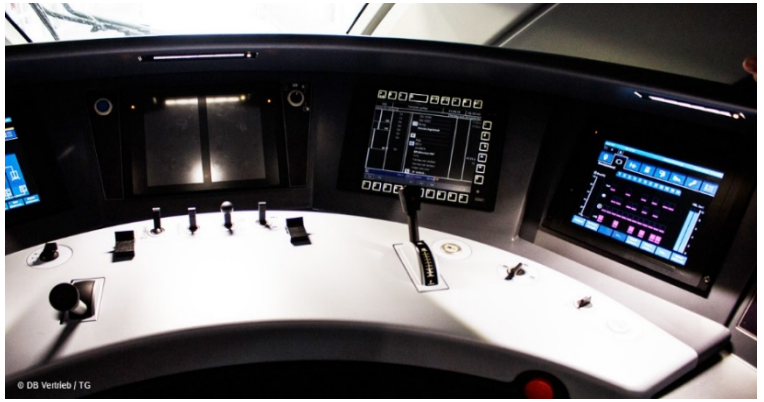
- Motivation and General Idea
- Introducing Example
- Implementation
- Experimental Evaluation
- Conclusions

Motivation

- Steadily increasing complexity
- ICs are widely used for safety-critical tasks
- Systems often exposed to environmental influences

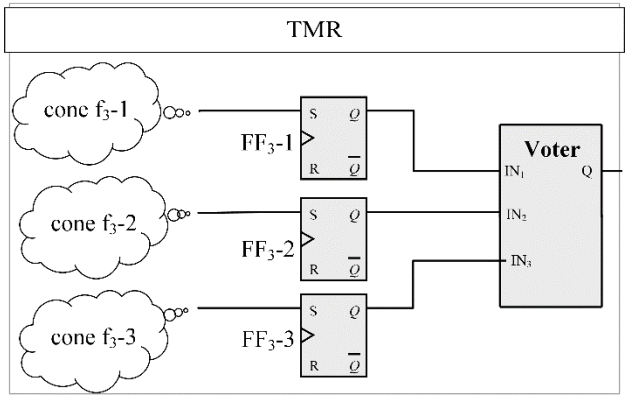


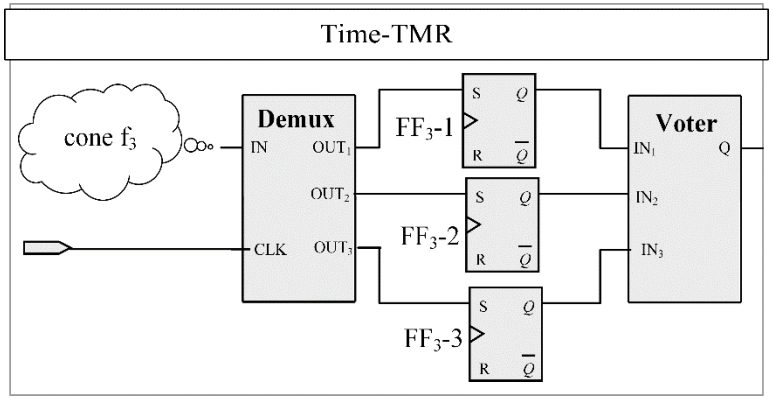




Motivation

- Different techniques exist to increase the robustness of integrated circuits
- Unfortunately these techniques lead to side-effects
  - Significant hardware overhead *(space-based)*
  - Strong influences on the timing behavior *(time-based)*
  - Not arbitrary applicable *(application-specific)*





General Idea

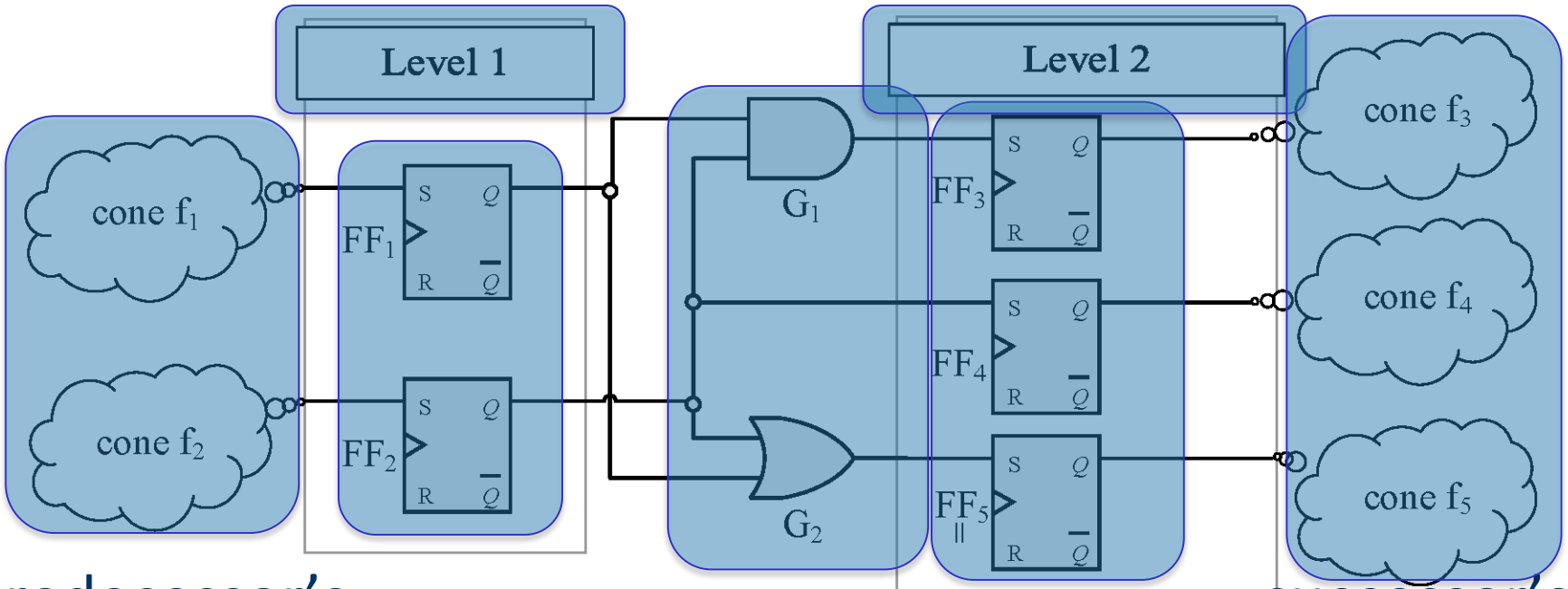
Developing an application-specific technique to enhance the robustness of arbitrary circuits

1. Orchestrating powerful formal techniques to investigate the design's behavior
2. Deriving properties that should hold under *fault-free* conditions
3. Realizing a mechanism that detects faults and rises a fault signal

Example

- Exemplary excerpt of a design containing

  2 logic gates 5 flip flops 2 hierarchical levels

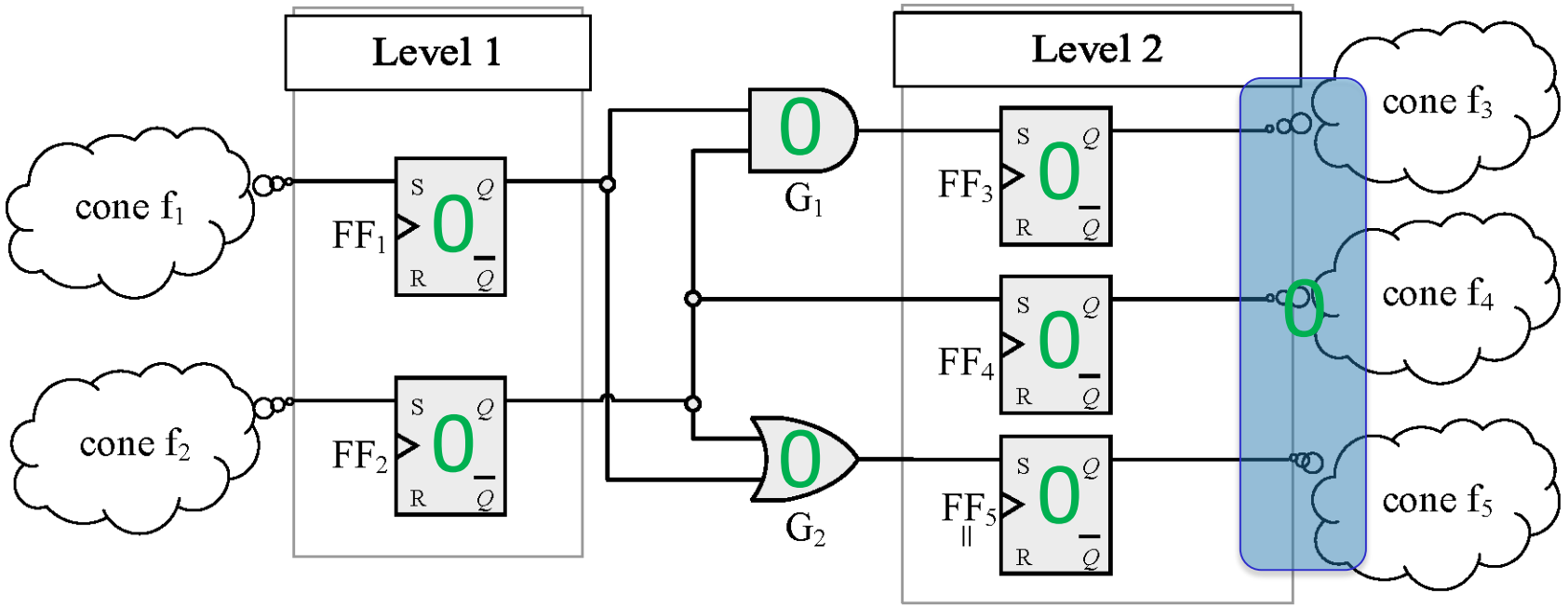


predecessor's cone

successor's cone

Example

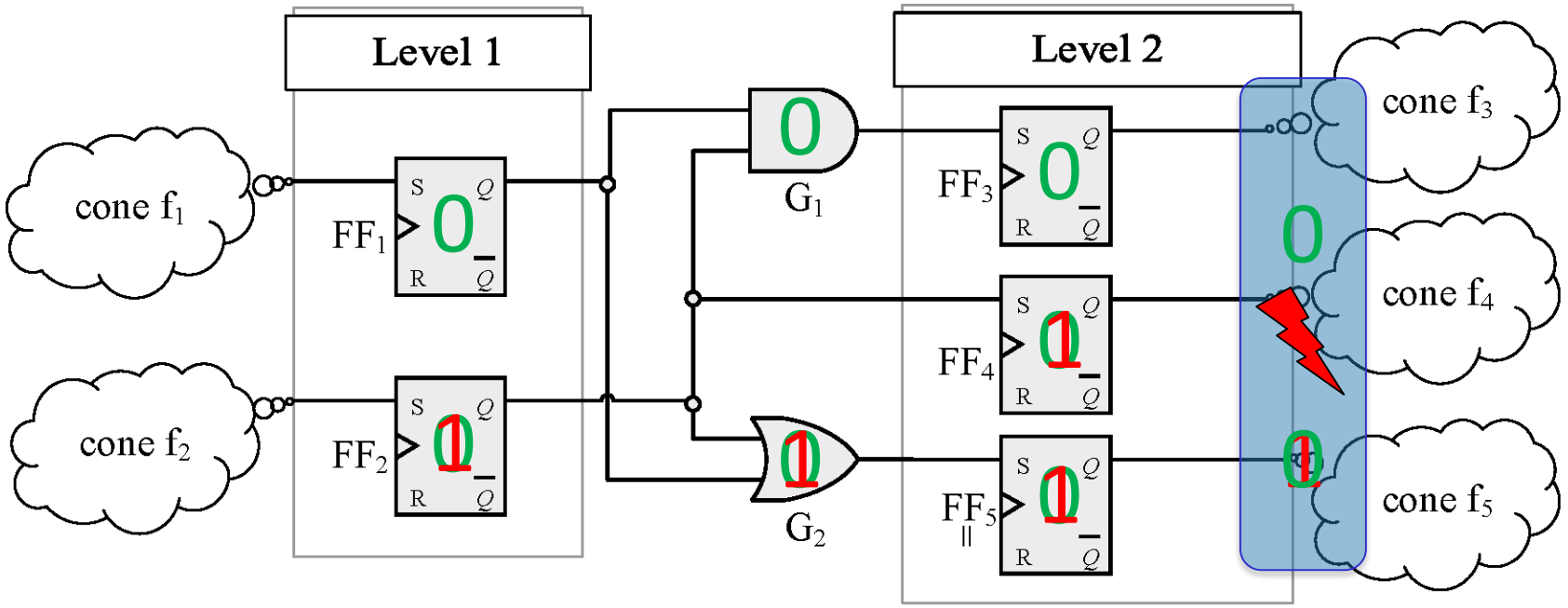
- Assume initial values for flip flops in level 1
- Results in values of logic gates and flip flops in level 2



Observation: FF_3, FF_4 and FF_5 have **same** output value!

Example

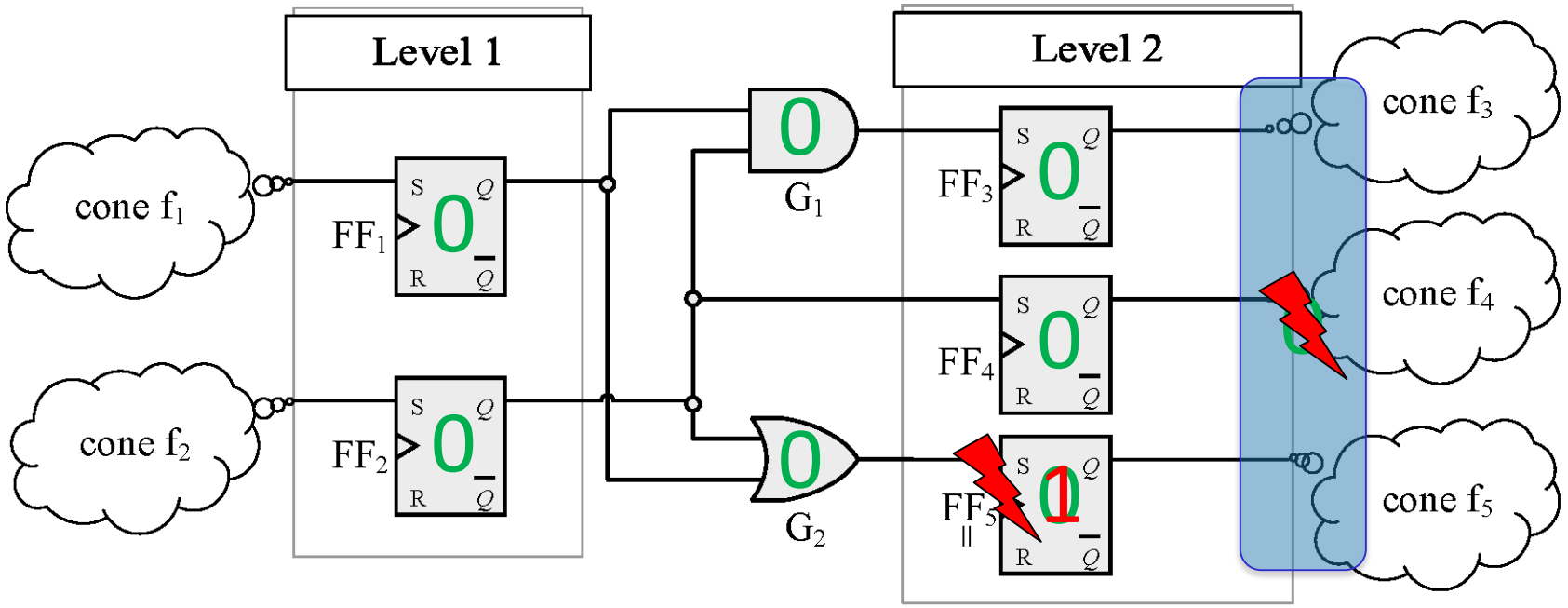
- Output value equivalence **depends** on level 1 values
- … and on a *suitable* partition of investigated FFs



Observation: FF_3, FF_4 and FF_5 have **different** output values!

Example

- Back to first scenario ➡ Equivalence holds
- Transient fault occurs at FF_5 ➡ Equivalence violated



Idea: This scenario detects an occurred single transient fault

Introduction

- Determine a *suitable* partition of flip flops **p**

  I. **A partition *p* of flip flops is a set of flip flops that are located in the same hierarchical circuit level.**

  II. **Suitable means: At least one state exists where the outputs of all flip flops in p *are* assumed equal.**

  ➢ Defined as the *Equivalence Property* (EP)

- Determine, store and compact such states

  ➢ Bounded Model Checking & Binary-decision diagrams

Introduction

- *Bounded Model Checking* (BMC)

$$\mathrm{BMC}(l) = I(s_0) \wedge \bigwedge_{0 \le i < l} T(s_i, s_{i+1}) \wedge P(s_l)$$

  - Initial state
  - Transition relation between two subsequent states
  - Property to be fulfilled in end state

- Adapted BMC to determine states

$$\mathrm{SFind}(P_j, l) = I(s_0) \wedge \bigwedge_{0 \le i < l} T(s_i, s_{i+1}) \wedge \mathrm{EP}(s_l, P_j)$$

Partition

Equivalence Property

Implementation

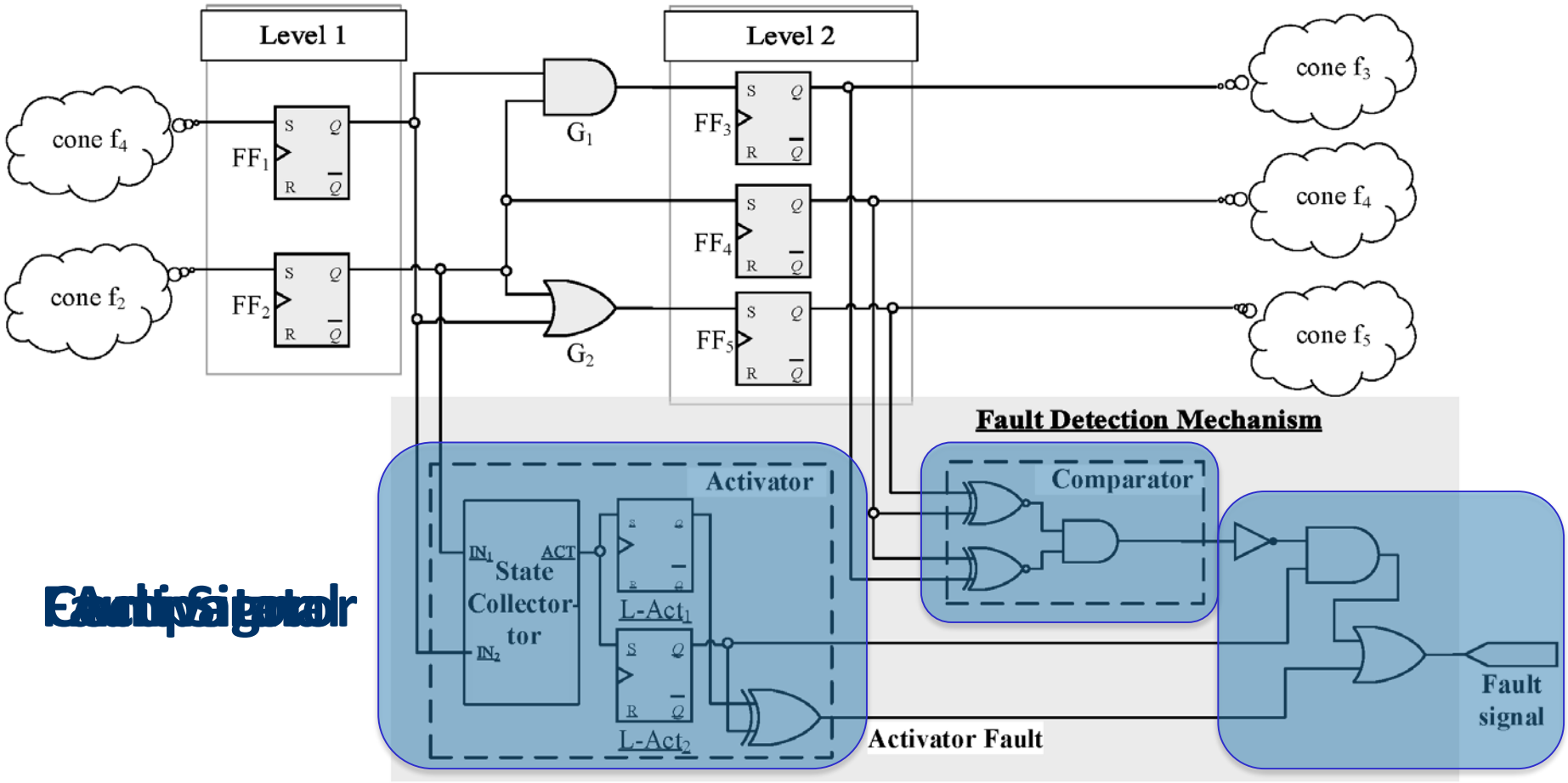
- Building Fault Detection Mechanism that consists of
  1. **Activator**

     Observes the current state of the circuit and decides whether it is possible to utilize it for fault detection.
  2. **Comparator**

     Evaluates if all flip flops in the current partition have the same output value: Check if EP holds
  3. **Fault Signal**

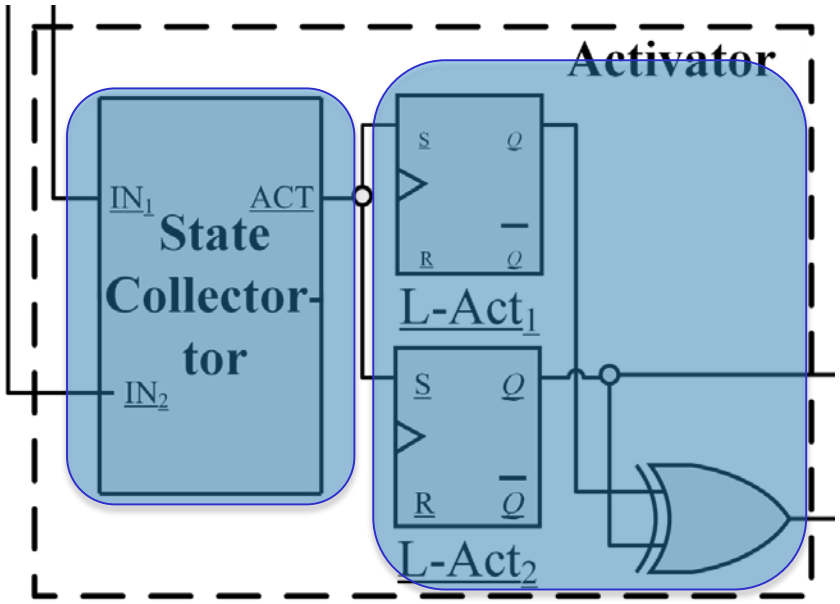
     Indicates whether a transient fault has occurred

Implementation



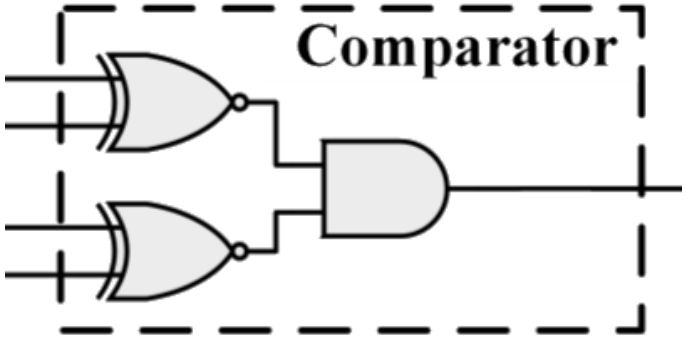
Implementation: Activator

- Stores suitable states in those an *Equivalence Property* holds
- States identified by formal analysis and stored in BDD
- Robust realization of latched output value



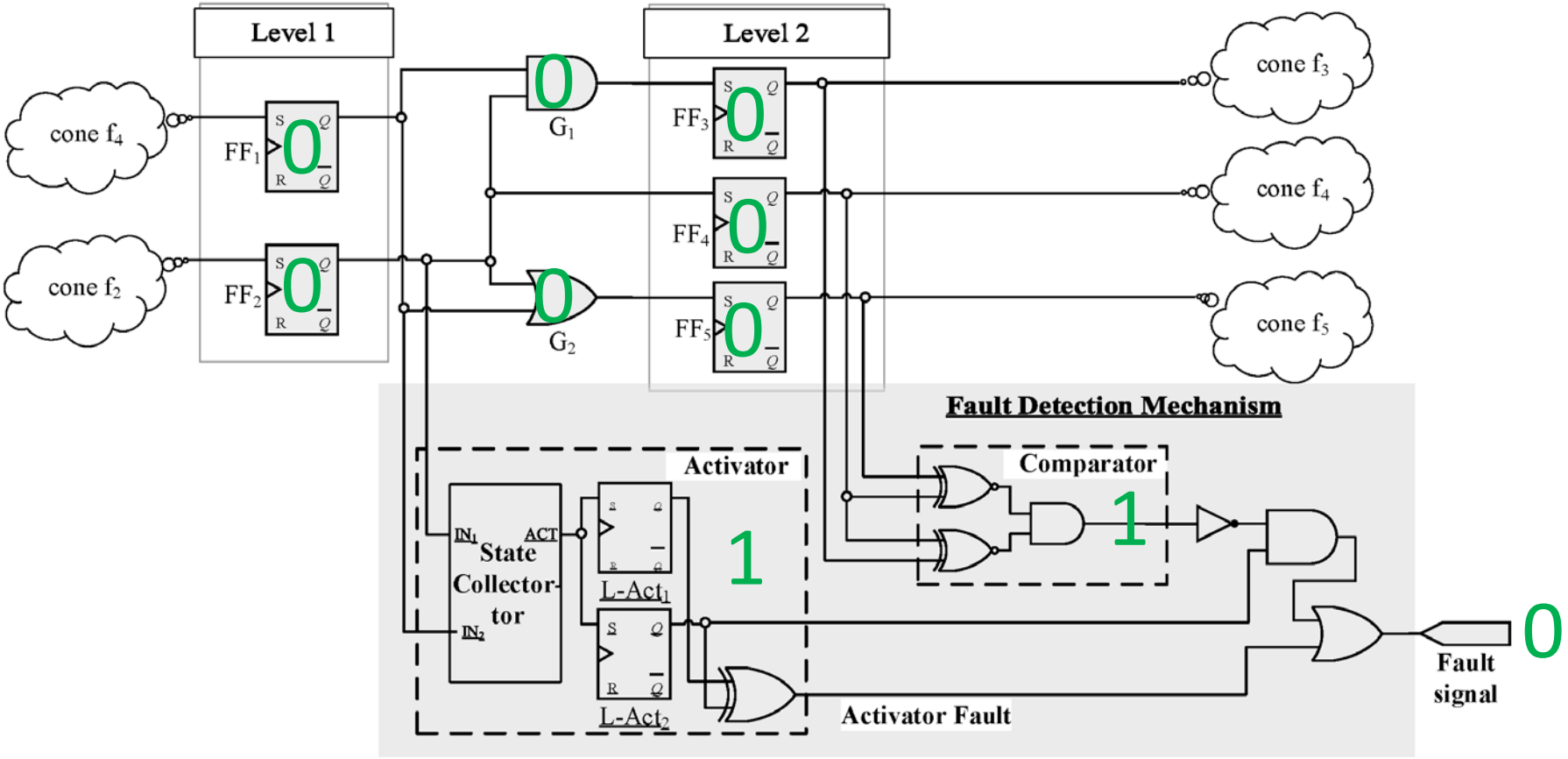
Implementation: Comparator

- Checks for equivalent inputs, i.e., flip flops output values
- Free scalable to *n* inputs (by cascading)
- Inputs connected to investigated flip flops (encodes partition elements)



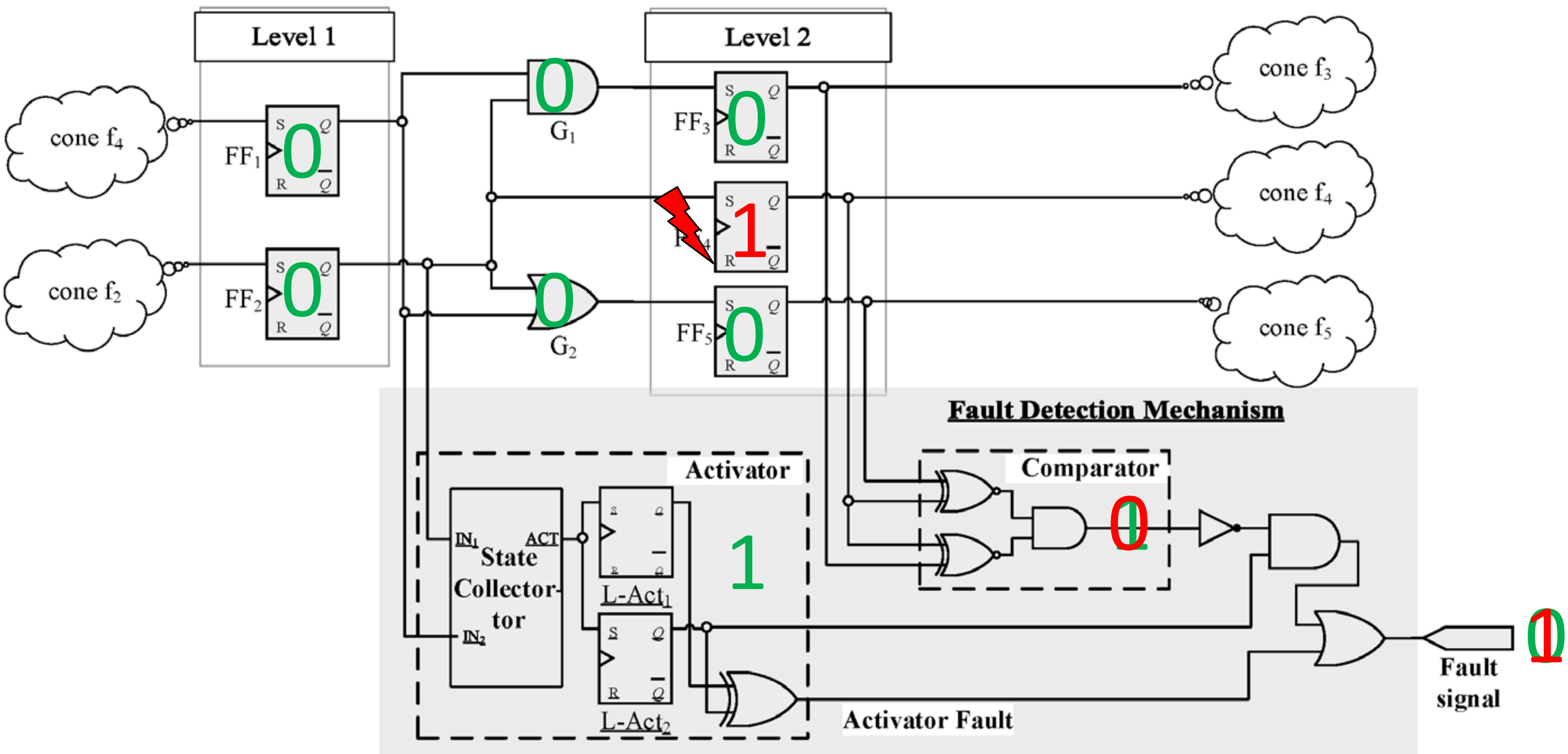
Implementation

- [illegible]



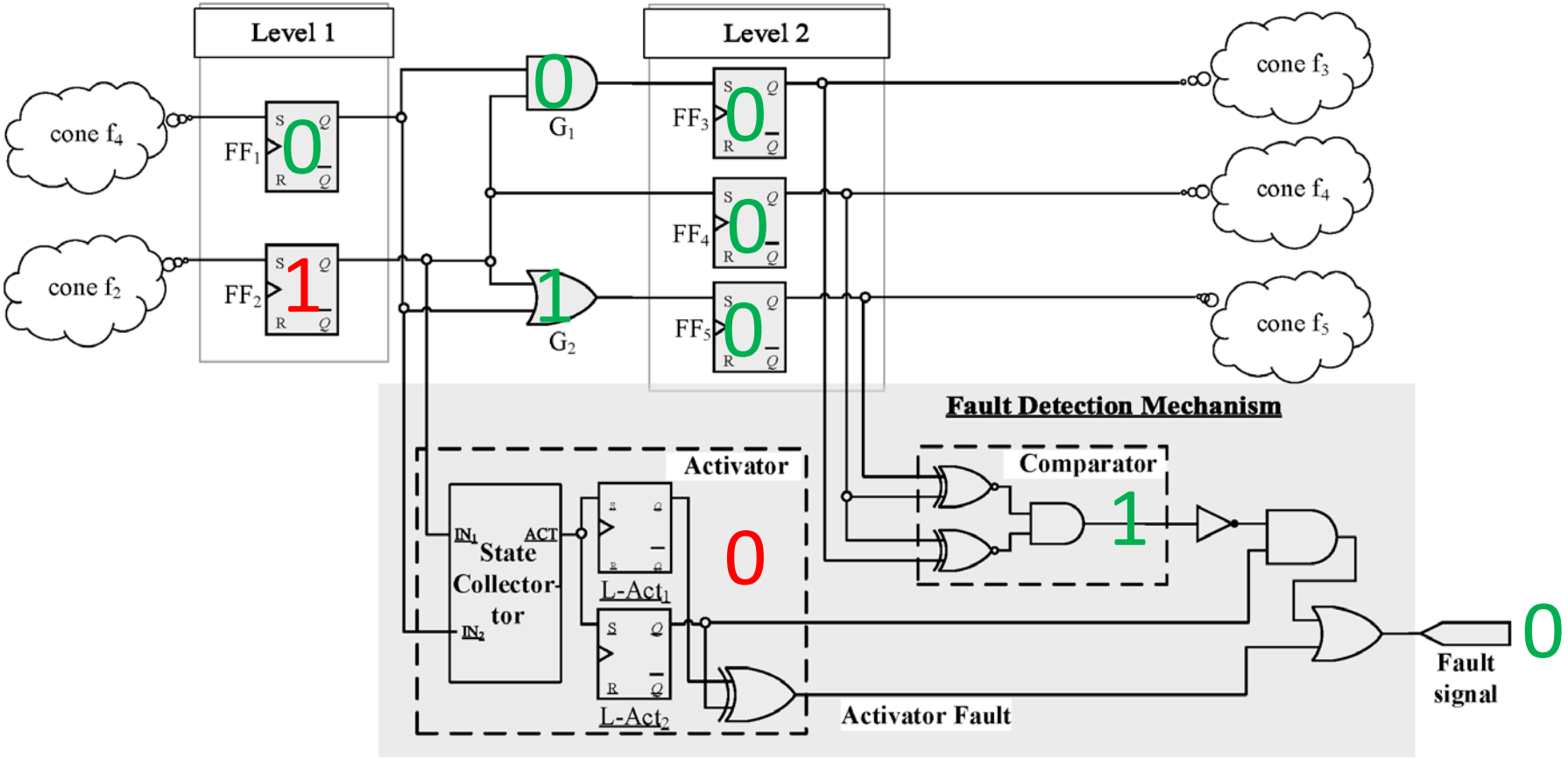
Implementation

- Transient fault in FF_4 / Fault detected: Fault Signal = 1



Implementation

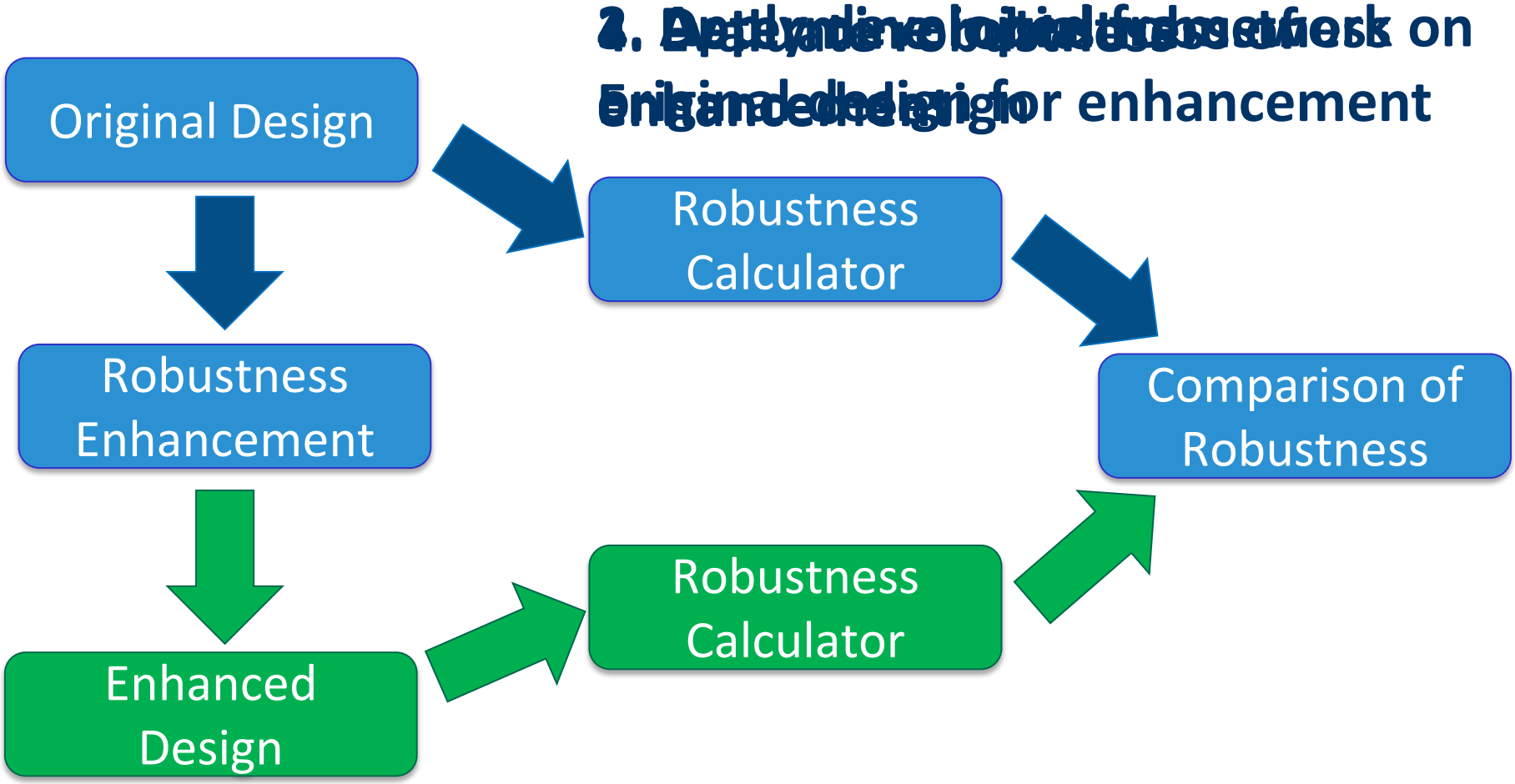
- Invalid state (Activator = 0) implies Fault Signal = 0



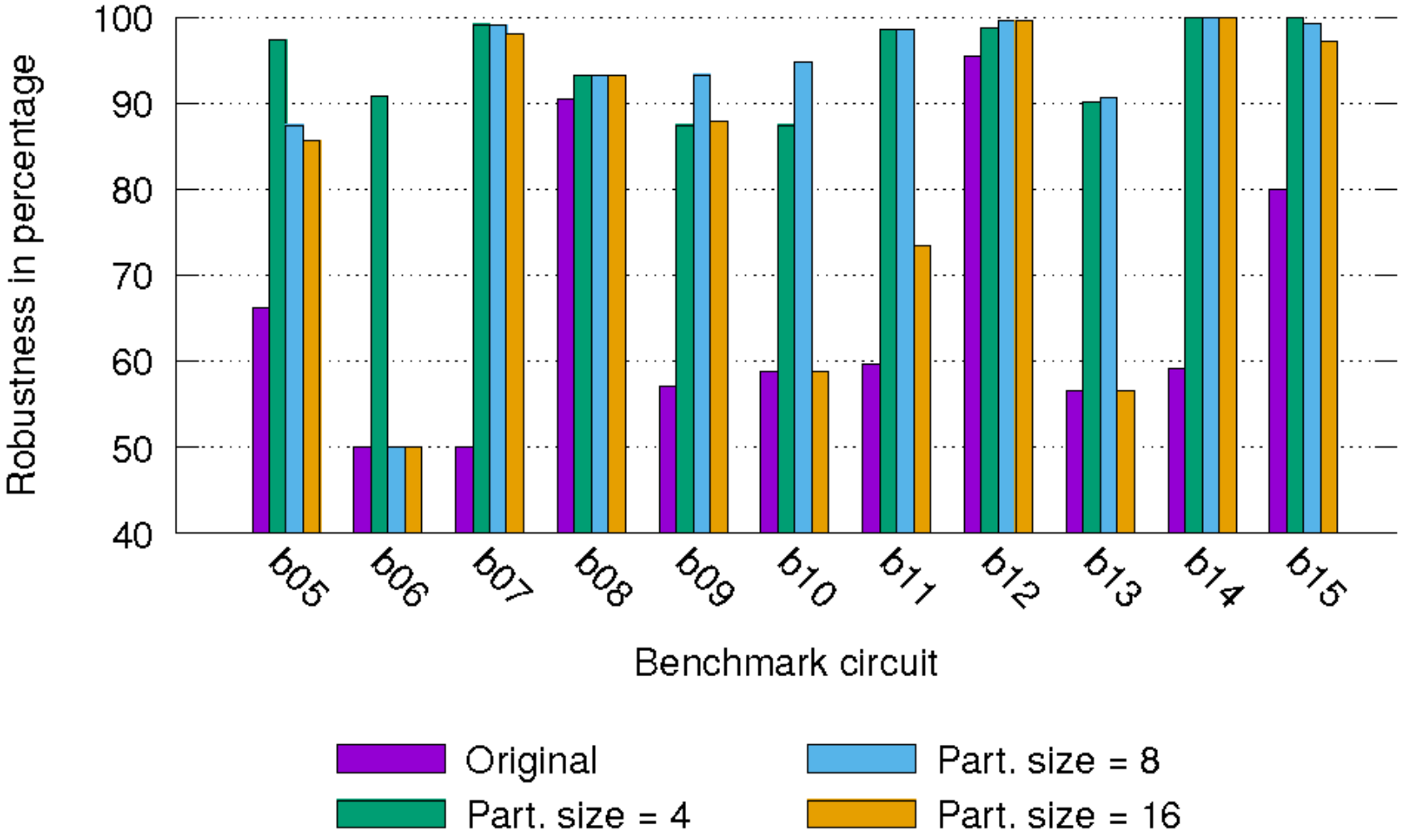
Experimental Evaluation

- Comparison between
  - Original designs (based on ITC'99)
  - Enhanced designs (different partition sizes)
- Original design processed by developed framework
  - Identifying Equivalence Properties
  - Realizing *Activator*, *Comparator* and *FDM*
- Simulation-based robustness calculator

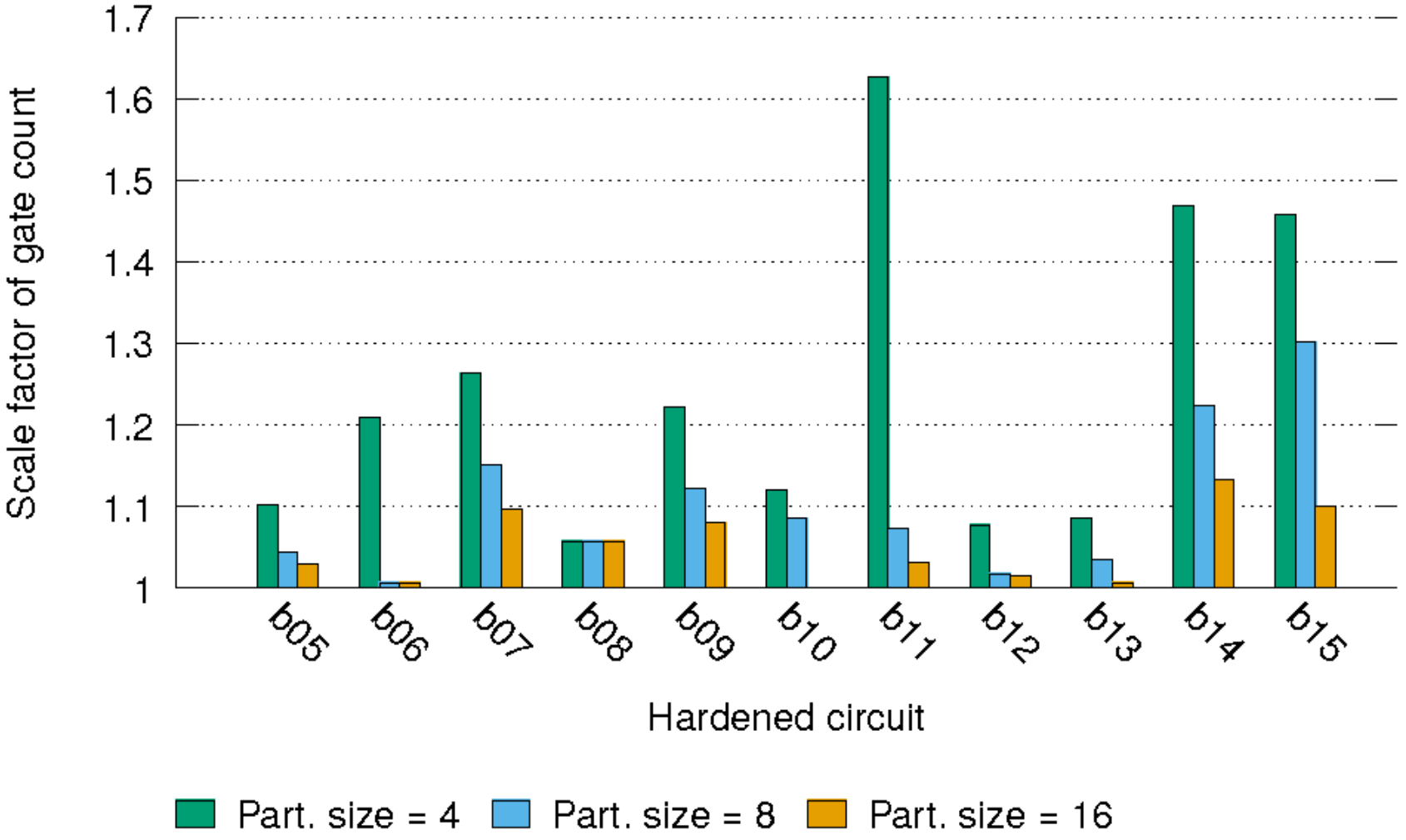
Experimental Evaluation



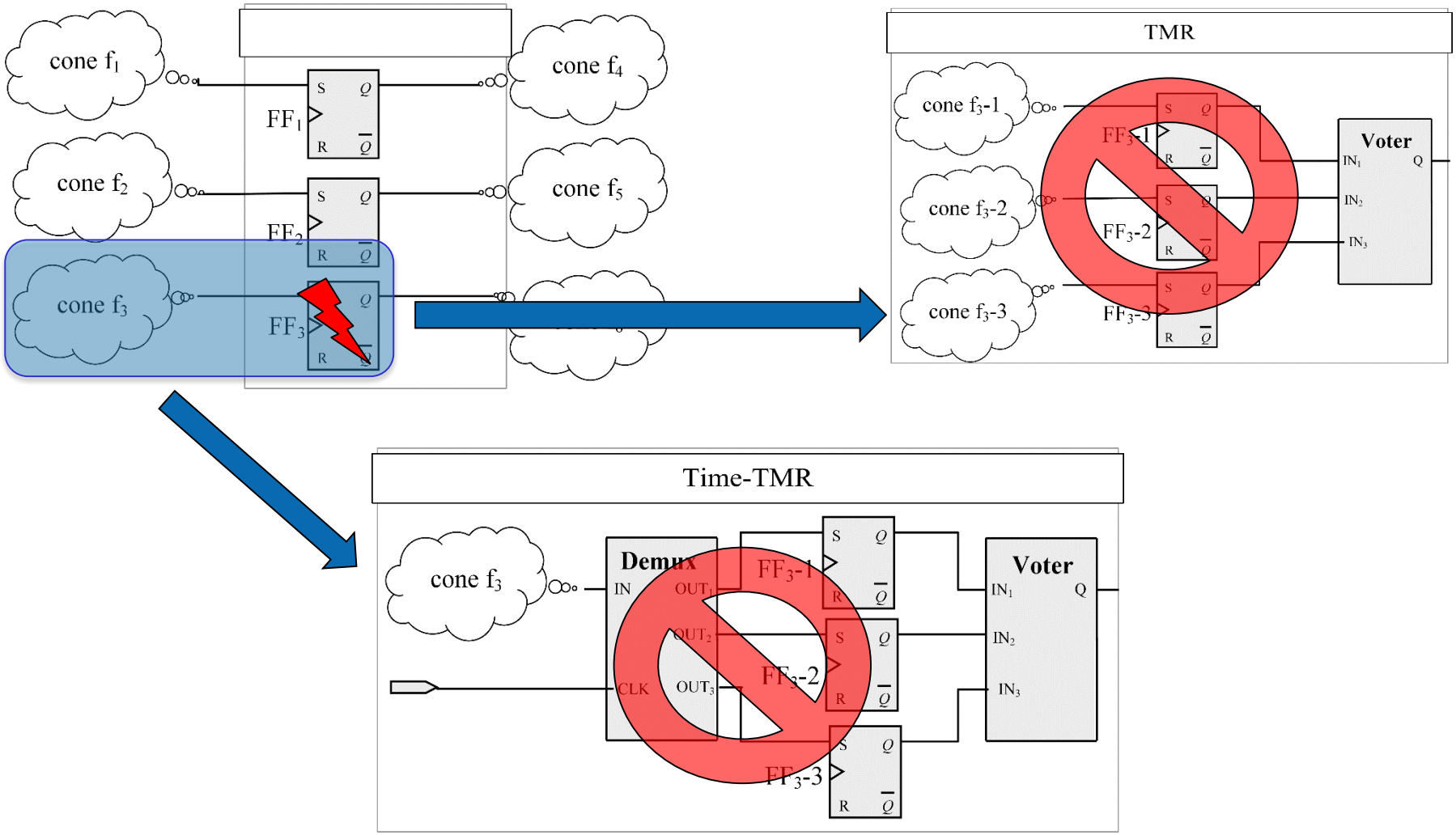
Robustness: original / enhanced design



Hardware Overhead



Conclusions



Conclusions

- Experiments show the potential of this approach
- Robustness enhancement to more than 90%
- Gate scaling factor by
  - 10.2 % to 62.7 % *(partition size: 4)*
  - 1.8 % to 30.2 % *(partition size: 8)*
  - 0.7 % to 13.4 % *(partition size: 16)*

- **Approach provides trade-off between robustness enhancement and manageable hardware overhead!**

Future Work

- Extension of the current procedure by a
  - Preprocessing step to determine promising partition sizes based on structural information
  - ➢ **Further improvement of robustness**
  - Mechanism for fault **correction** by using determined application-specific knowledge
  - ➢ **Addressing new field of applications**

Pictorial Sources

- Oracle Sparc M7 Architectural View
  http://www.enterprisetech.com/wp-content/uploads/2014/08/oracle-sparc-m7-die-shot.jpg

- ICE4 Control Unit
  https://inside.bahn.de/wordpress/uploads/2016/09/F%C3%BChrerstand_ICE4.jpg

- DLR: SmallGeo Satellit
  http://www.dlr.de/rd/en/Portaldata/1/Resources/portal_news/newsarchiv2007/smallgeo_front.jpg

Thanks for your attention!

Enhancing Robustness of Sequential Circuits Using Application-specific Know-ledge and Formal Methods

Sebastian Huhn

Stefan Frehse

Robert Wille

Rolf Drechsler

Universität Bremen

JOHANNES KEPLER UNIVERSITÄT LINZ

Appendix

- Benchmarks

circ.	#gates	#FFs	run time [s]		
			$p_s = 4$	$p_s = 8$	$p_s = 16$
b05	608	66	7.71	1.42	1.42
b06	66	9	0.11	< 0.10	< 0.10
b07	382	51	34.83	10.78	10.77
b08	168	21	0.23	0.23	0.23
b09	131	28	1.61	0.66	0.66
b10	172	17	3.95	1.05	1.50
b11	366	30	60.80	1.02	0.46
b12	1000	121	238.62	75.35	69.06
b13	309	53	16.29	5.40	6.65
b14	3461	247	1287.13	341.21	105.38
b15	6931	447	28787.10	5115.23	917.77

Appendix

- State Collecting procedure

Algorithm 2: State Collecting procedure

Data: enumerated partition: P_j, max. number of states: u
Data: unrolling depth: l

1 $\widehat{S} = \emptyset$ /* stored as BDD */
2 **for** $k = 1$ **to** l **do**
3 $\quad F = \mathsf{SFind}(P_j, k)$
4 $\quad$ **repeat**
5 $\quad\quad$ **if** $|\widehat{S}| > u$ **then return** $\widehat{S}$
6 $\quad\quad \widehat{S} = \widehat{S} \cup s_{i+1}$ /* collects state */
7 $\quad\quad F = F \wedge \neg s_{i+1}$ /* blocks solution */
8 $\quad$ **until** $\mathsf{SAT}(F)$
9 **return** $\widehat{S}$

Appendix

- Partition Enumeration procedure

Algorithm 1: Partition Enumeration procedure

Data: set of non-robust FFs: N_i, upper-bound partition size: p_s
Ensure: $0 < p_s \leq |N_i|$

```
1  Container E = ∅        /* Data container for EPs */
2  while p_s > 1 do
3      Let P_j ∈ P(N_L) such that |P_j| = p_s
4      if P_j = ∅ then
5          p_s = p_s − 1
6          continue
7      Ŝ = StateCollector(P_j)
8      if Ŝ ≠ ∅ then
9          E = E ∪ EP(Ŝ, P_j)
10         N_i = N_i \ P_j
11     else N_i = N_i \ {f} with f ∈ N_i (chosen after analysis)
```