Soft-Error-Analysis

Patrick Klampfl

patrick.klampfl@student.tugraz.at

Computer Science

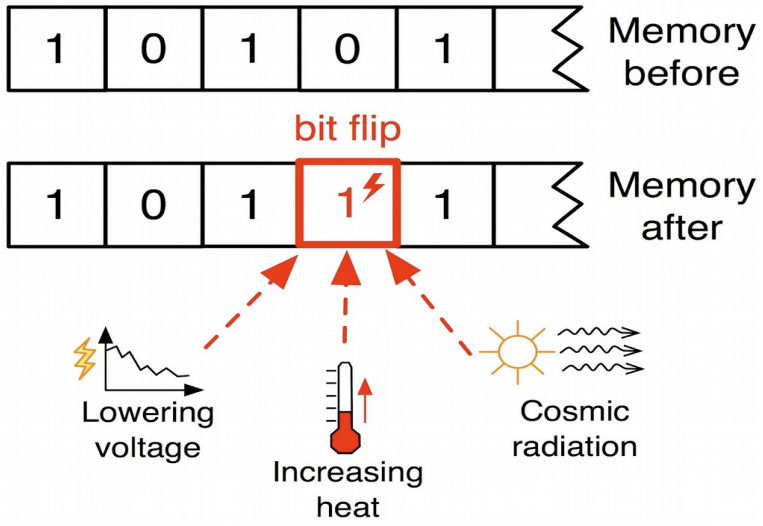
September 15, 2015

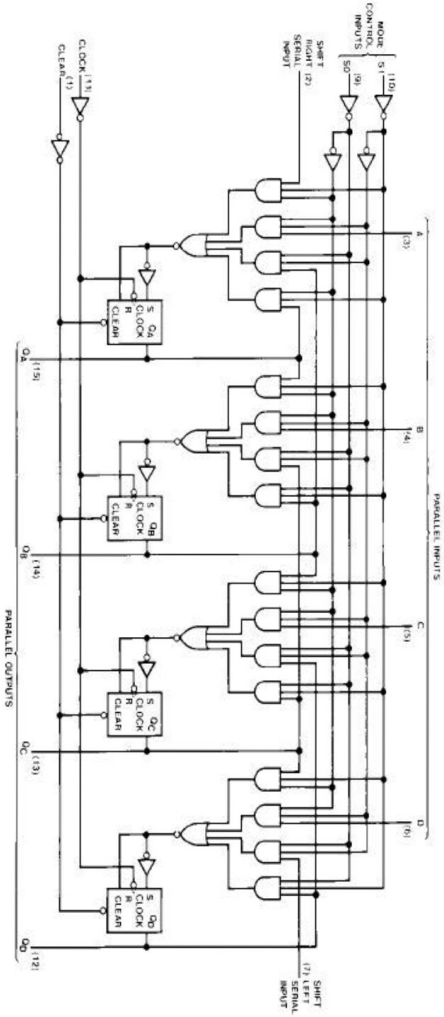
Overview

- Introduction: Soft-Errors & Soft-Error Analysis
- Detect Soft-Errors
- Verify protection Logic (main part)
  - input format, algorithms, output format, ..
- First results
- Conclusion / future work

Soft Errors

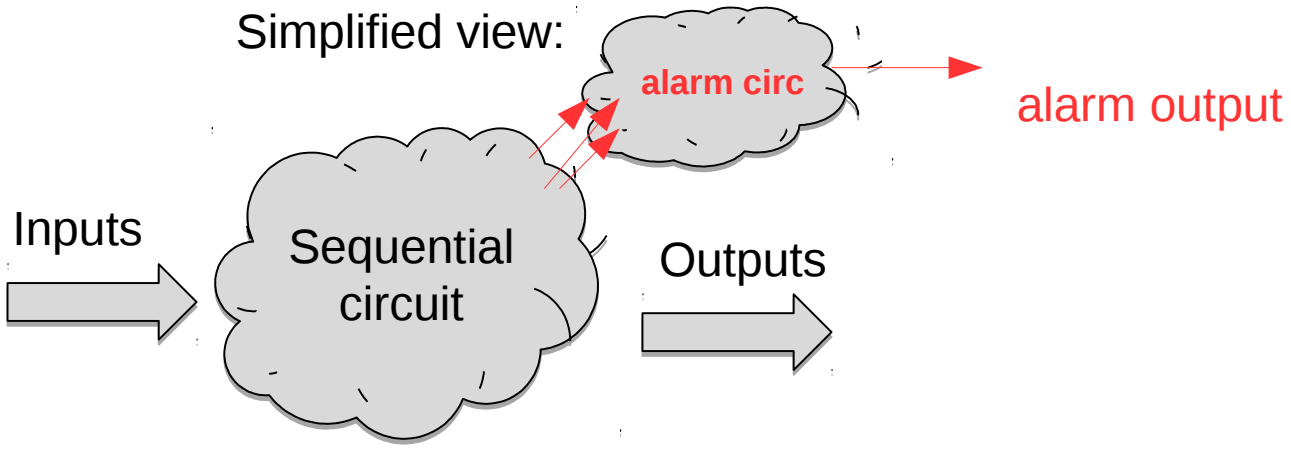
- Boolean circuits: inputs, AND gates, latches, outputs
- Components (latches, AND gates) can have soft-errors
  - flip truth value
  - Single fault assumption (only one component flips)



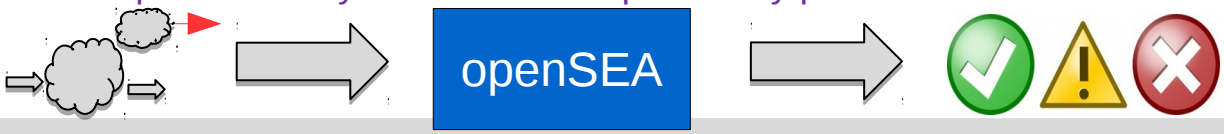


Soft-Error-Analysis

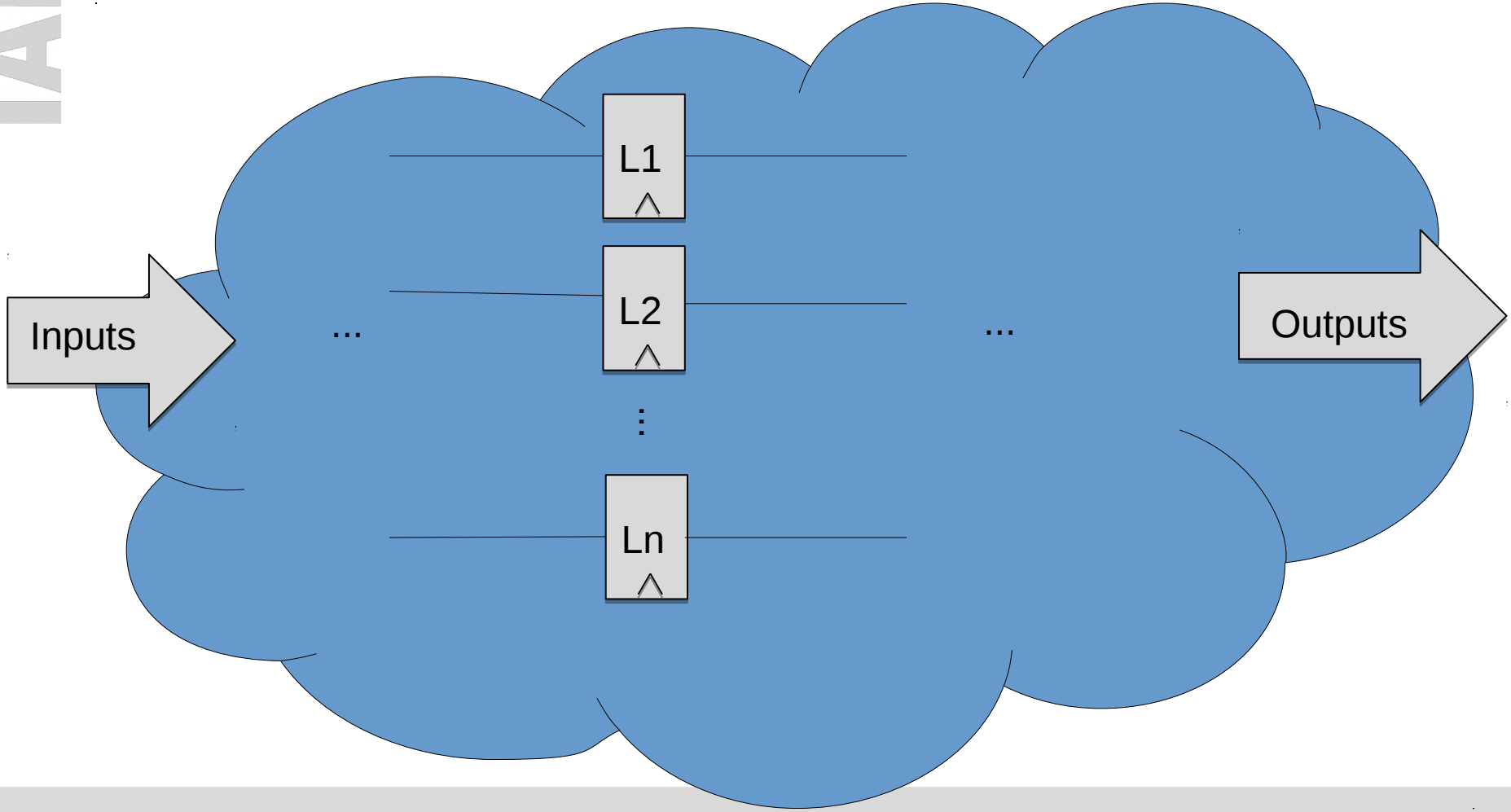
- Circuits to detect soft-errors aka protection Circuit:
  - Alarm output: true, when flip has an effect on the value of the outputs



- Main goal of my work: create Tool that tests the soft-error-detection for completeness (semi-formally ;-):
  - Report definitely vulnerable and potentially protected circuits

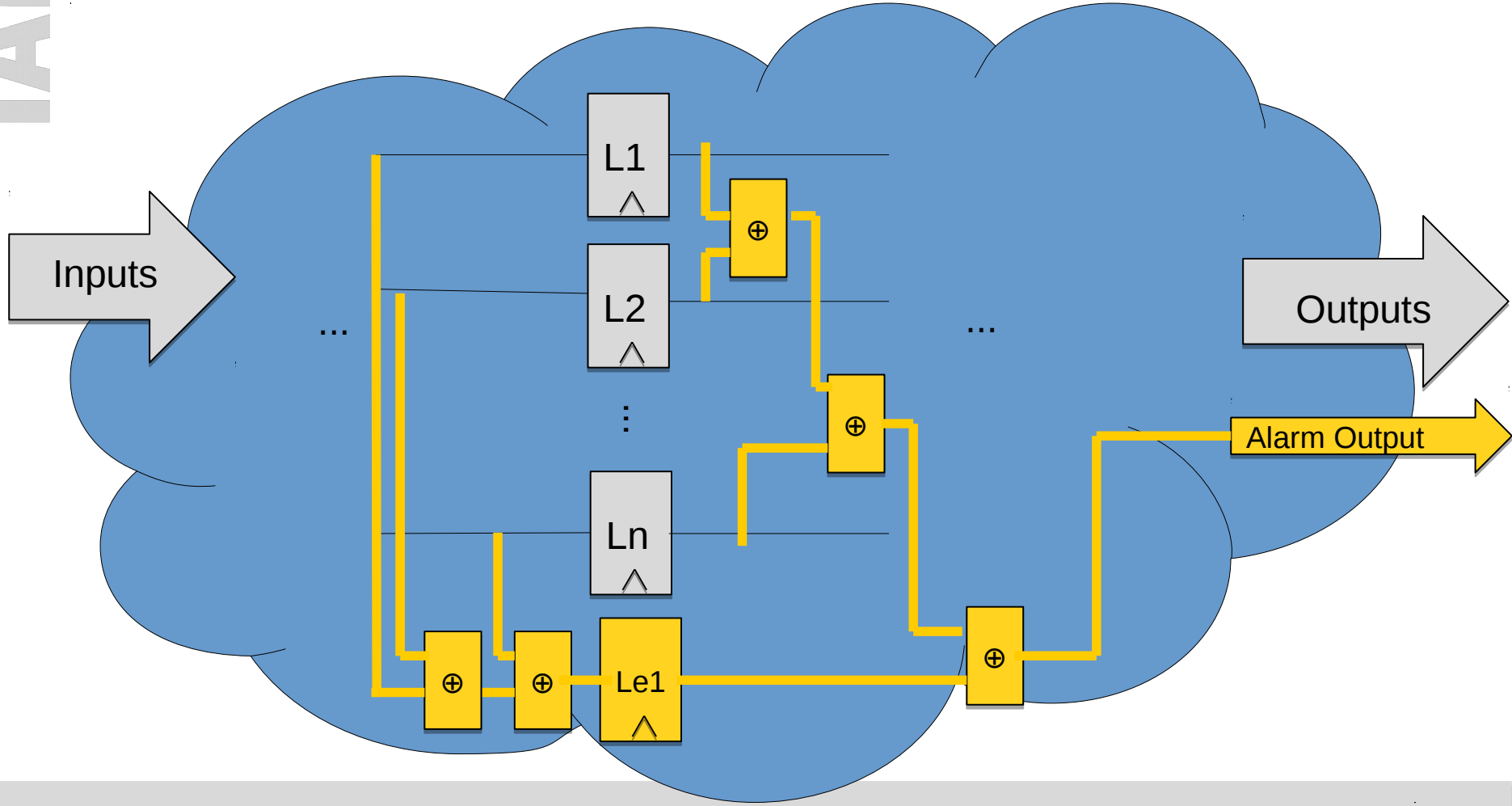


How to detect Soft-Errors?

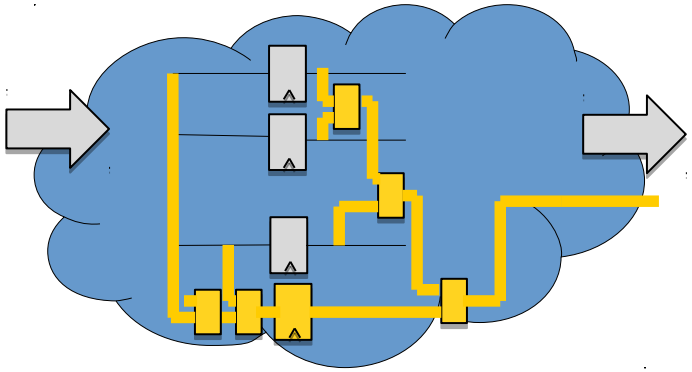


How to detect Soft-Errors:

- → add **redundancy**! New Tool: **AddParityTool**



How to detect Soft-Errors:

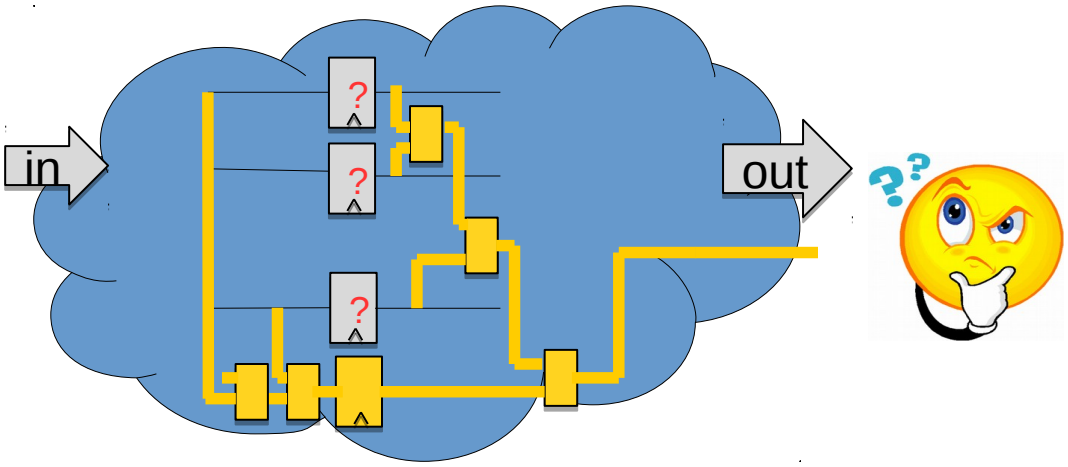


- **AddparityTool:**
- Input:
  - Circuit to protect (in AIGER format)
  - Percentage of latches to protect
  - New latch per how many existing latches (k)
    - Relevant for critical circuit depth
- Output:
  - Protected circuit
  - With new output: **alarm**

  - Contains k extra latch(es)

Is the protection-circuit correct ... ?

- ... or is there some scenario where a latch can be **flipped without recognizing** that?
- which latches are **definitely** vulnerable?
- which are potentially protected?



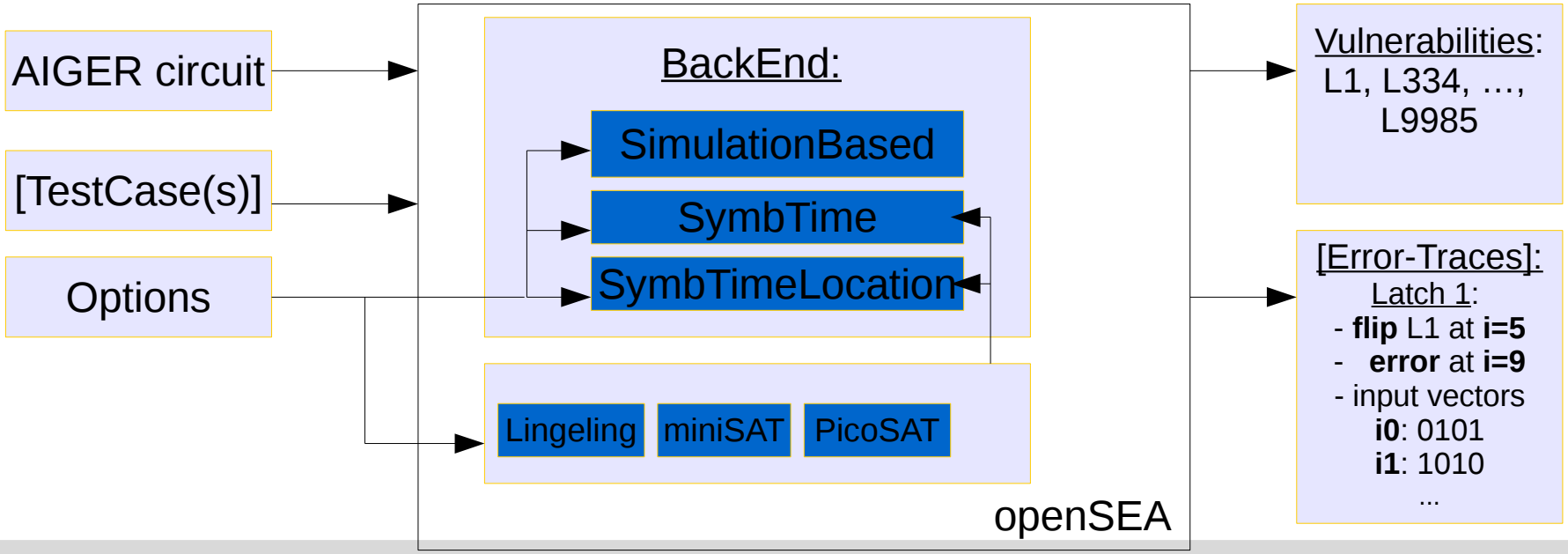
Is the protection-circuit correct ... ?

- ... or is there some scenario where a latch can be **flipped without recognizing** that?
- which latches are **definitely** vulnerable?
- which are potentially protected?
- The openSEA (tool) can help you with these questions!
  - open **S**oft-**E**rror-**A**nalysis – a tool to verify protection circuits (working title)

openSEA

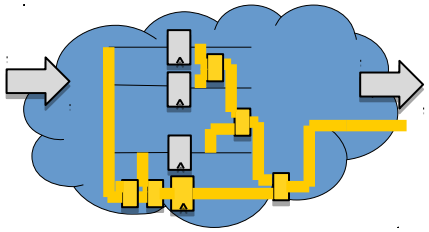
- Input: arbitrary circuit with protection logic (alarm output)
- Output: List of definitely vulnerable latches

- highly modular, extensible and configurable
- several algorithms, modes, options, optimizations, ...



Input and Output Format

- Input Format:
  - circuit with protection logic



  - TestCase(s) consisting of a vector of input values

time	Input 0	Input 1
ts 0	1	1
ts 1	0	1
ts 2	1	0

Input and Output Format

- Output Format:
  - List of definitely vulnerable Latches

L3
L42
L1337

  - Optional: ErrorTraces for each vulnerable Latch (to stdout or to file)
    - When happens the flip, when has it an effect on the output, what were the necessary inputs?

L3:	Flip at ts 10		
	Wrong output at ts 13		
	time	Input 0	Input 1
	ts 0	1	1
	ts 1	0	1
	ts 2	1	0

BackEnd: SimulationBasedAnalysis

(1) Execute **correct simulation** with the provided TestCase (vector of input-vectors)

store the resulting output vectors

(2) Compare with **all possible faulty simulations**

for each latch:

for each time-step: flip truth value of latch (= timestep i)

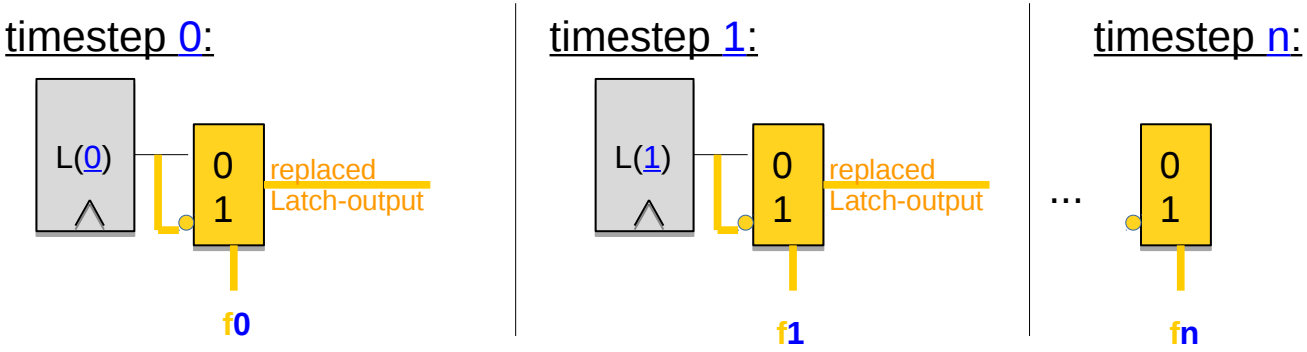
vulnerable, if output value is different in this or in a future-time-step (= timestep j)

Implementational Details:

- Implemented AIGER circuit simulator for that

BackEnd: SymbTimeAnalysis (1)

Idea: make **point in time** where a latch value is **flipped** symbolic, call SAT-Solver



- **Do for each latch:**
- Convert circuit to a CNF transition relation (Tseitin transformation)
- Unroll transition relation **for each timestep** in the TestCase
  - replace input variables with concrete input values from TestCase
  - append modified copy each iteration, previous state to next state
  - add clause(s) saying that the output has changed
  - call SAT solver

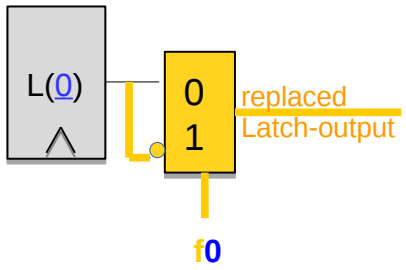
BackEnd: SymbTimeAnalysis (2)

- If SATISFIABLE:

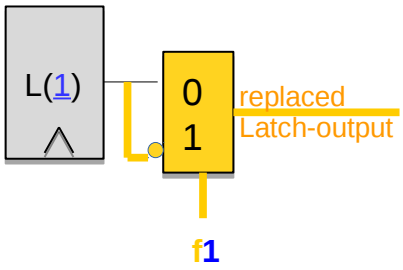
  assignment: ¬f0, **f1**, ¬f2, .. , ¬fn

  - current latch is vulnerable
  - SAT assignment: read point in time where a flip creates an error (f variable)

timestep 0:



timestep 1:



timestep n:

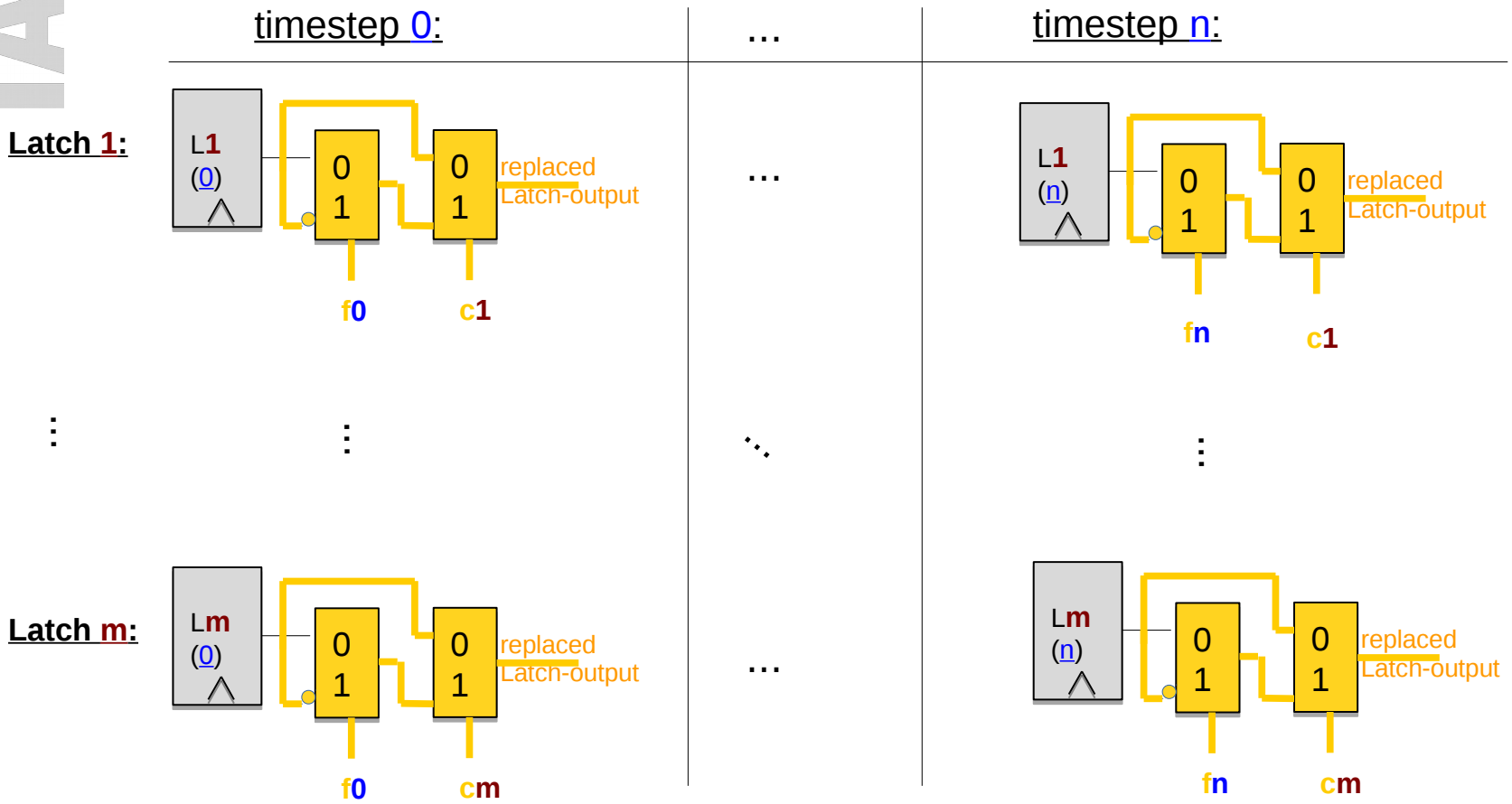
...

Implementational Details:

- Incremental SAT solving
- mode: don't copy whole transition-relation, build CNF on-the-fly

BackEnd: SymbTimeLocationAnalysis (1)

Idea: make **point in time** AND the **latch to flip** symbolic



BackEnd: SymbTimeLocationAnalysis (2)

- If SATISFIABLE:
  - Found vulnerability
  - SAT assignment:

assignment: ¬f0, **f1**, ¬f2, .. , ¬fn, ¬c1, ¬c2, **c3**, .., ¬cm

- c3: Latch 3 is flipped
- f1: it is flipped in timestep 1

BackEnd: SymbTimeLocationAnalysis (3)

- Unroll transition relation **for each timestep** in the TestCase:
  - append modified copy each iteration, previous state to next state
  - replace input variables with concrete input values from TestCase
  - clause(s) saying that output has changed

Extension: TestCase with free Input-Values

- allows TestCases to have undefined input-variables
- For SymbTimeAnalysis and SymbTimeLocationAnalysis
- Not (easily and efficiently) possible for SimulationBasedAnalysis
- SAT-solver chooses the values for inputs to force an error

- Example TestCase with free inputs

time	Input 0	Input 1
ts 0	1	1
ts 1	1	?
ts 2	?	0

…

& Example Error-Trace

L3:

flip at ts 10

Wrong output at ts 13

time	Input 0	Input 1
ts 0	1	1
ts 1	1	1
ts 2	0	0

…

Extension: TestCase with free Input-Values

- Get the best of both worlds:
  - Discrete input values when possible (faster!)
  - Variable input values when necessary (flexible)
- This makes OpenSEA a full model-checker!
  - With detailed feedback for the hardware designer where the bug is

time	Input 0	Input 1
ts 0	?	?
ts 1	?	?
ts 2	?	?

…

L3:

flip at ts 10

Wrong output at ts 13

time	Input 0	Input 1
ts 0	1	0
ts 1	0	1
ts 2	0	1

…

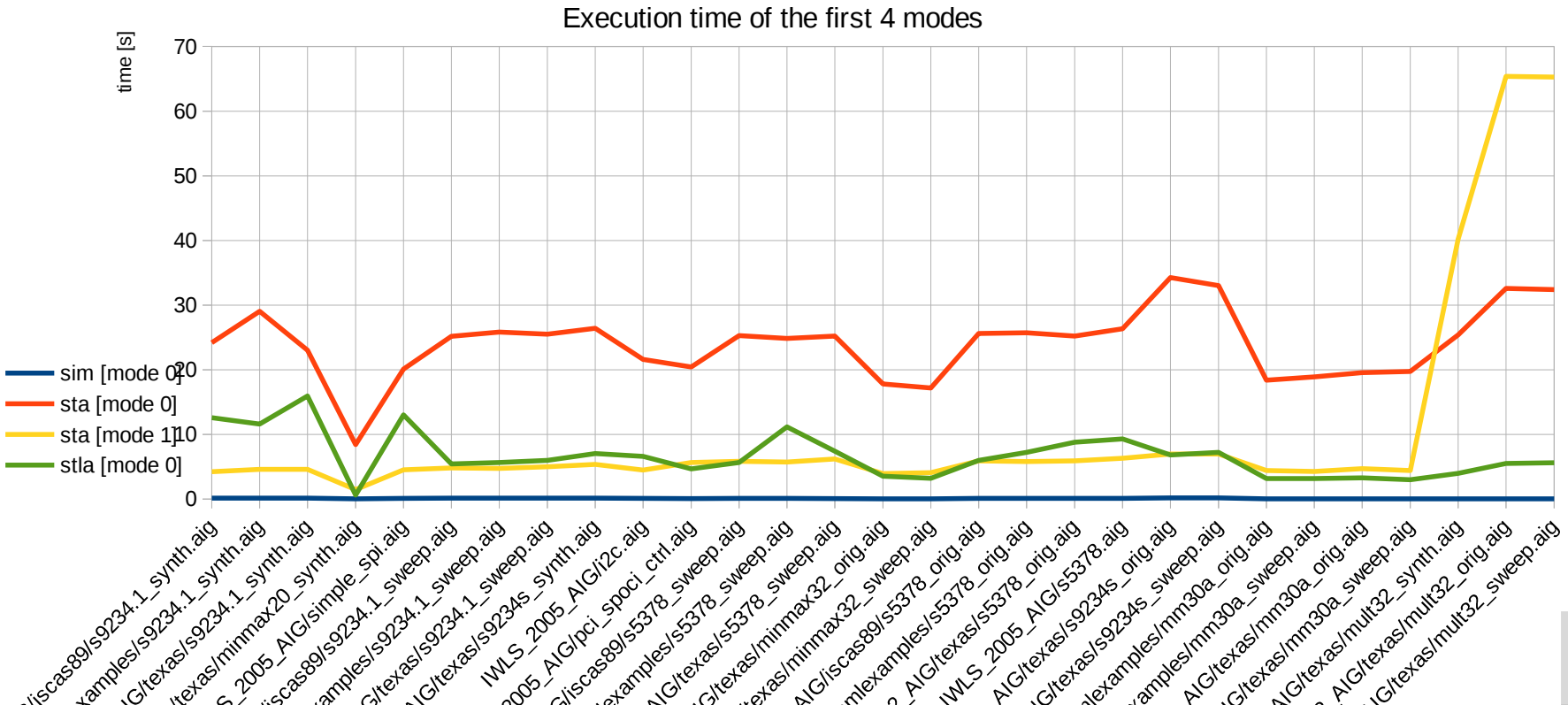
What else have I done?

- Convert circuit with protection logic to a model-checking problem (standalone tool):

```
// ------------------------------------------------------------------------------------------
// @brief creates 2 copies of original_circuit to translate it to a MC problem:
// error if the outputs of the 2 circuits are different, but alarm is set to false
//
//                              +---------+ outputs1:
//          i1  +-----+--------+          +-------+
//          i2  |-----|--+-----| circuit |       |
//          i3  +-----|--|---+-+         +----+  |    +-------+
//                    |  |   | +---------+    |  +----+       |
//                    |  |   |                +-------+       |          +----+
//                    |  |   |              outputs2:  |       +----------+    |
//                    |  |   | +---------+    +-------+  !=   |          |AND +-----+ bad
//                    |  |   +-+         +----+       |       |     +---o|    |
//                    |  +-----| circ_cpy|       +----+       |     |    |    |
//                   +--------| (modif) +-------+    |       |     |    +----+
//          f_in+-----[magic]--+-+-------+-----+      +-------+     |
//                             |               |                    |
//        ci .. cn -------------+               +-------------[magic]-+
//                                                 alarm_output
//
//
//
aiger* aiger_create_MC_copy(aiger* original_circuit)
```

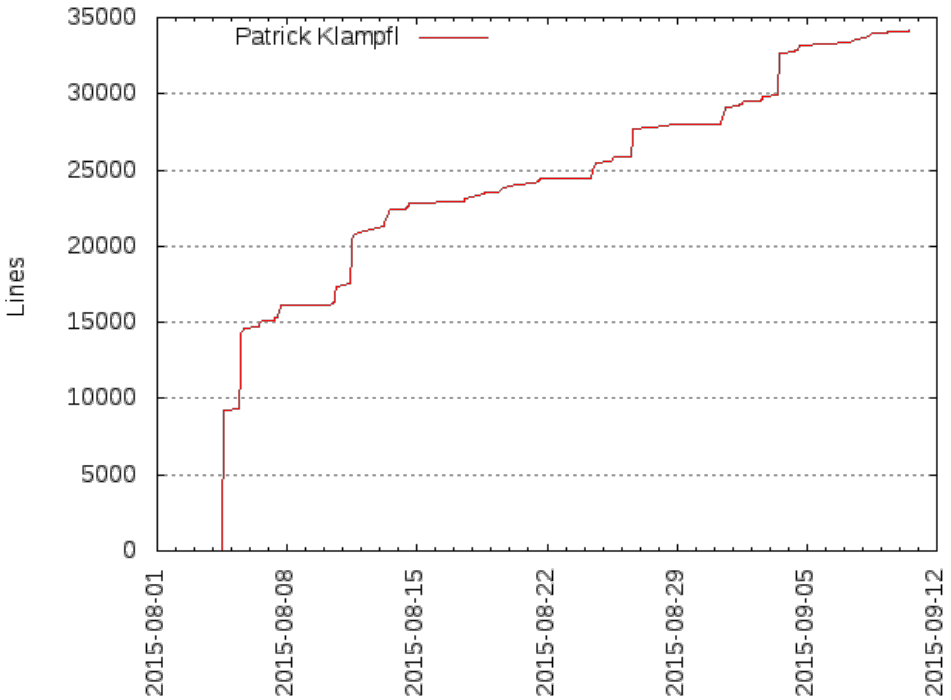
What else have I done?

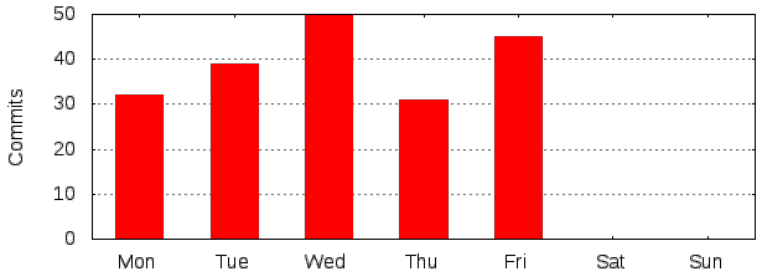
- Start running some benchmarks on the developed algorithms with several modes, optimizations, etc., on problem-instances



What else have I done?

- A lot of coding



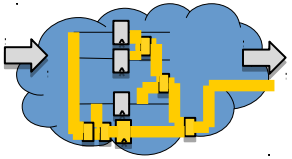


What else have I done?

- drinking Coffee ;-)

Conclusion

- AddParityTool: adds simple parity computation to protect a circuit



- Create configurable SEA Tool: **openSEA**
- Implement Soft-Error-Analysis Algorithms:
  - Simulation-Based Analysis
  - SAT-Based: Symbolic Time
  - SAT-Based: Symbolic Time + symbolic Location

- Optimizations, Free Inputs extension
- Implement pure Model-Checking Approach as well
- First Benchmarks

Future Work

- more benchmarking
- quantitative analysis: how likely is each vulnerability
- Check other components for vulnerabilities:
  - Besides of latches, AND gates could be flipped as well
- Detect false alarms
- Model an environment and add exceptions:
  - output might not be relevant at each timestep
- … and lots of other things you can think of

Questions?

Thank you for your attention!