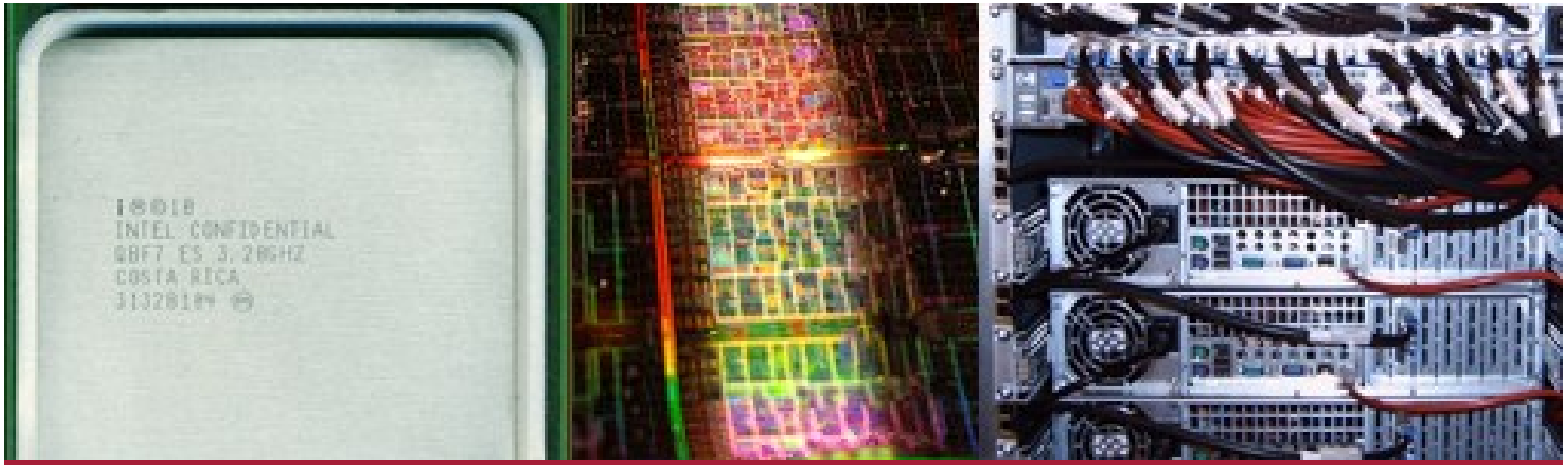
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT



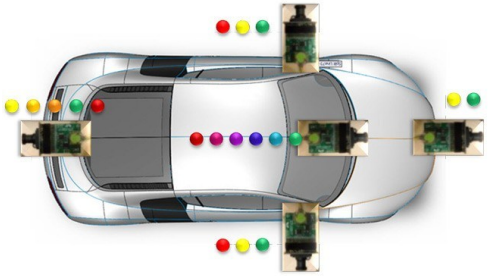
SMoSi: A Framework for the Derivation of Sleep Mode Traces from RTL Simulations

Dustin Peterson and Oliver Bringmann
Eberhard Karls Universität Tübingen, Germany

27 January 2016

What is my talk about?

Wearables[1]



Driver Assistance[2]

Smartphones[3]

Power Optimization of Embedded System Designs

[1] http://www.technikblog.ch/2014/10/pulsmessung-am-handgelenk-kann-das-ueberhaupt-genau-sein/
[2] http://www.xilinx.com/applications/automotive.html
[3] http://marketingland.com/report-us-smartphone-penetration-now-75-percent-117746

Low Power Optimization in SoC Designs

- Lots of well-known power reduction techniques available!
  - Clock gating, power gating, DVFS, …
  - Each requires partitioning a design into power domains.

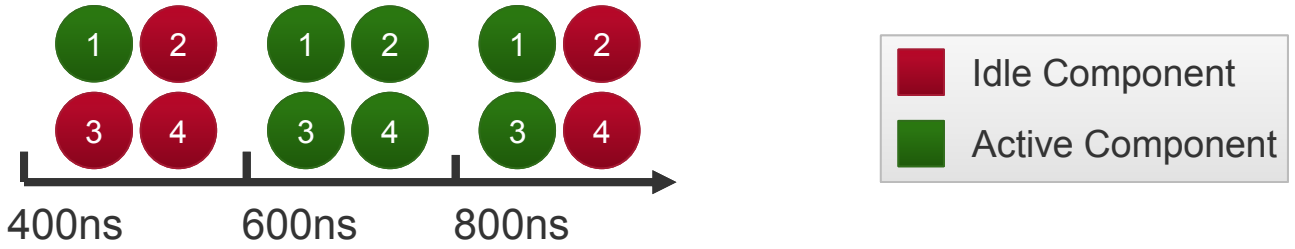
- Optimality is highly **application-specific**!

- Approach for application-specific partitioning:
  e.g. [*Dobryal et al., Workload driven power domain partitioning]*
  - Application model is represented by a **sleep mode trace.**

Sleep Mode Traces

- A **sleep mode trace** assigns each time frame in a simulation run a set of idle components.

- **Example:**



How to obtain a sleep mode trace?

- Manually
- Randomly
- High-level models (e.g. Real Time Calculus[1])

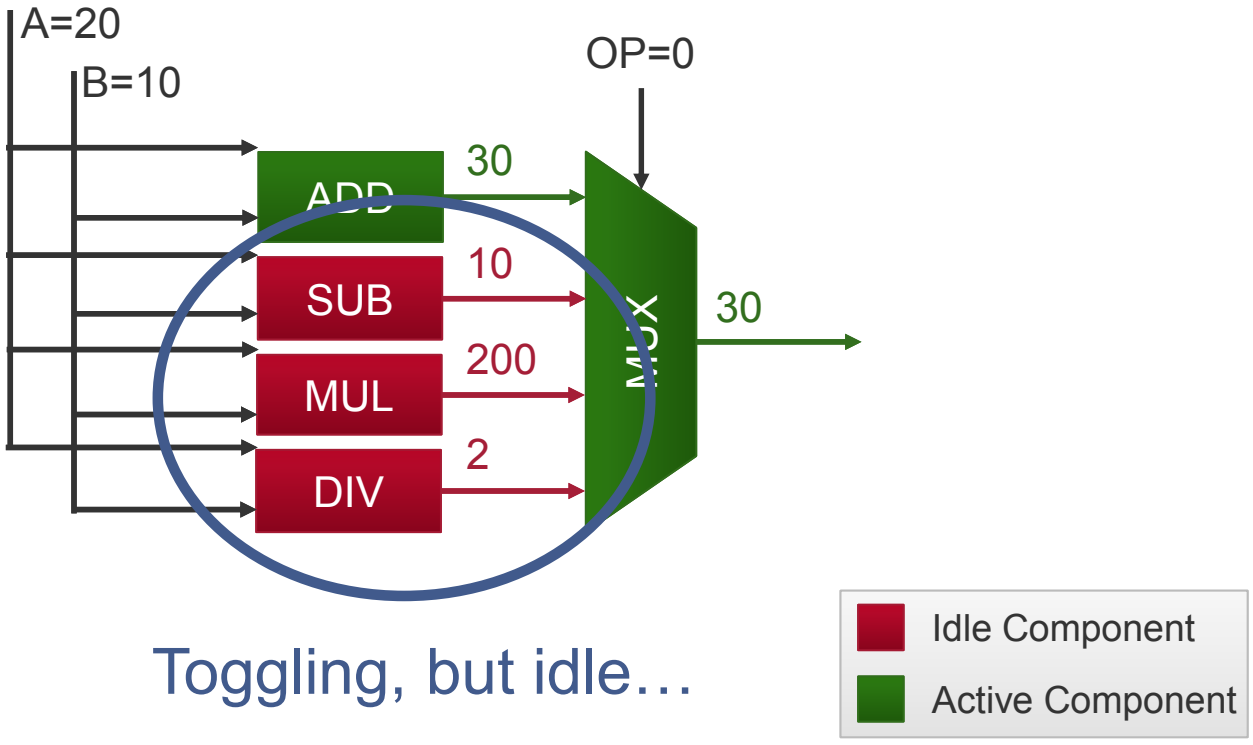
- **Can we generate it at RTL?**

[1] Chakraborty et al. A general framework for analysing system properties in platform-based embedded system designs

Obtaining Sleep Mode Traces at RTL

- Perform RTL simulation and check in the resulting waveform, if there is activity in a component?

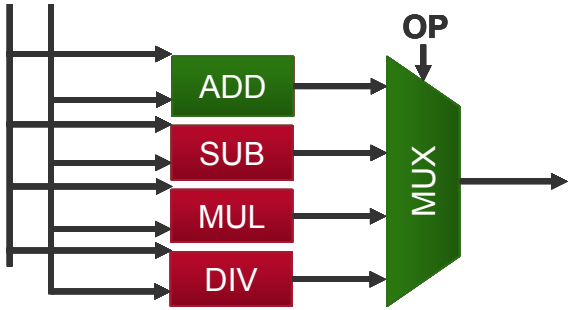
- **Not working!** Control structures may cut off redundant datapaths!



Basic Idea of SMoSi

- **Idea:** A priori generation of rules (**Idle Rule Set**)
  - *„Under which condition can a component be set idle?“*
  - Application of these rules to the waveforms (VCD)

Example Design



Idle Component Active Component

Idle Rule Set

Comp.	When idle?
ADD	$OP \neq 0$
SUB	$OP \neq 1$
MUL	$OP \neq 2$
DIV	$OP \neq 3$
MUX	$never$

Waveforms

Time	OP
10ns	1
20ns	3
30ns	1
40ns	0

Agenda

(1) **A Priori Rule Generation in SMoSi**
Sleep Mode Simulation in detail

(2) **Experimental Results**
SMoSi under test with open-source designs

(3) **Conclusion and Future Work**
Advancement and applications of SMoSi

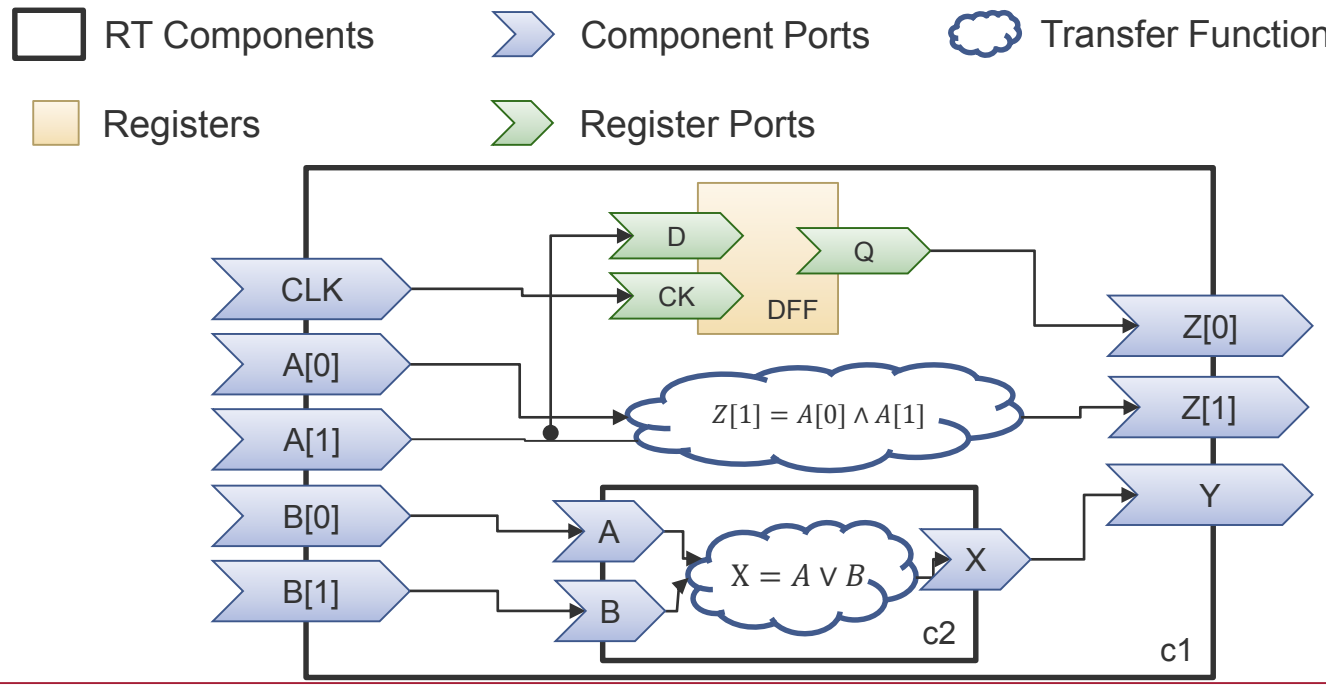
Sleep Mode Simulation (SMoSi) in detail

1 A Priori Rule Generation in SMoSi

A Priori Rule Generation
Design Graph

- **Goal:** Formal analysis of a design to get component-level idle rules
- **First step:** Synthesize RTL design to netlist with **exact mapping.**
  - Netlist is already elaborated and mapped to boolean transfer functions!
  - But it has **same components**, **registers** and **ports** as RTL design!

- Netlist is parsed and converted into a **design graph**:

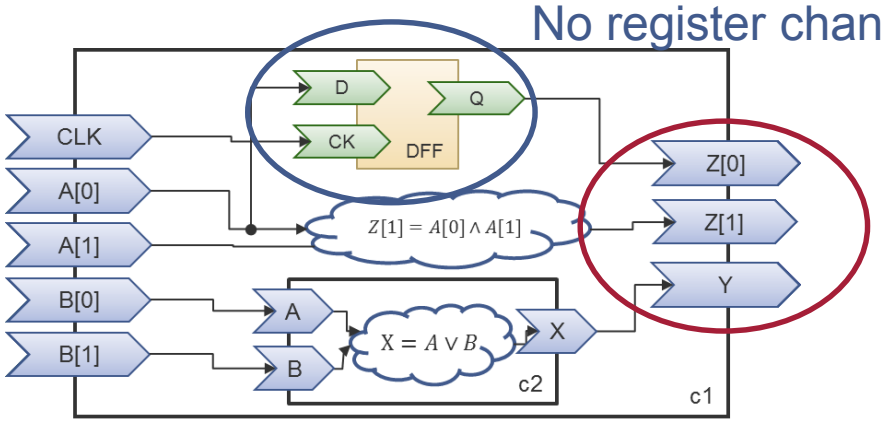


A Priori Rule Generation
From the Design Graph to the Idle Rule Set

1. Build port-level **redundancy rules** using Shannon Expansion!
   - *„Port X is not used under condition Y.“*
   - Applies to component and register ports!

2. From the port-level rules we derive component-level idle rules!

A component is idle, if all of its **output ports** are redundant in each transfer function!
Furthermore, the **component‘s state** must not change!

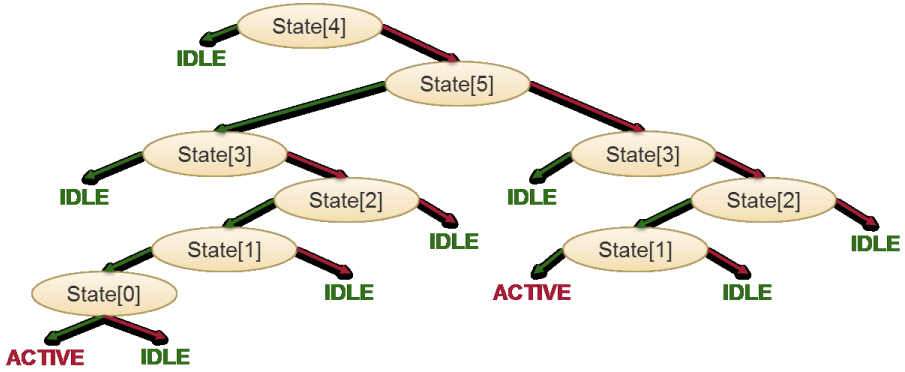


A Priori Rule Generation
From the Design Graph to the Idle Rule Set

1. Building the **idle rule for the output interface**:
   - The port-level redundancy rules of all output ports must evaluate to true!

2. Building the **idle rules for the registers**:
   Analyze port-level redundancies of the register input ports (data input, clock, enable) of all registers!

Result: BDD that is representing the full idle rule of a component!



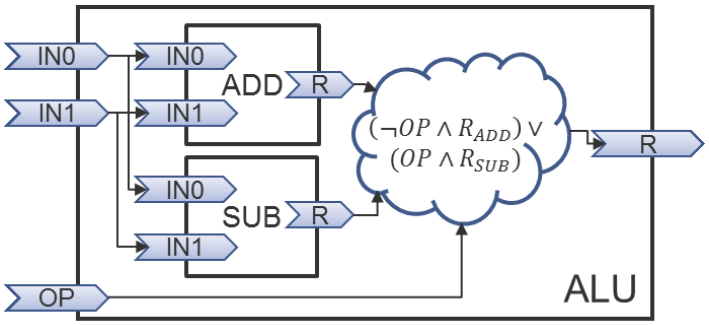
Idle Rule for the ALU component in an 8080-compatible processor

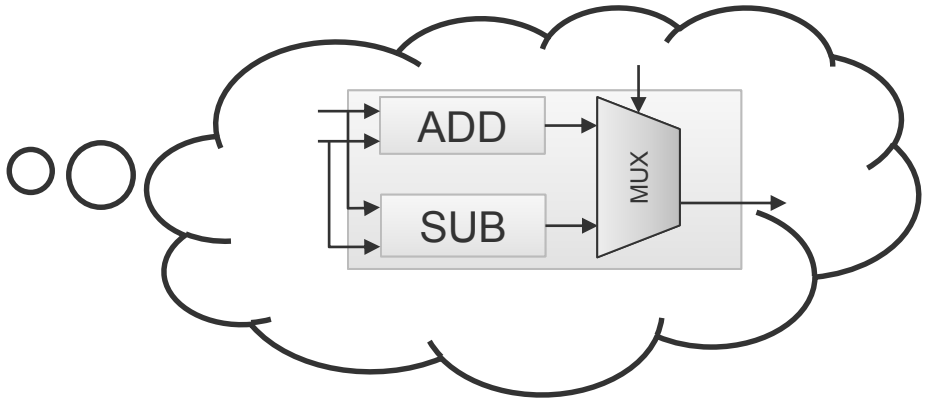
A Priori Rule Generation

How to derive port-level redundancy rules?

- Derive **redundancy rules** for each port using *Shannon Expansion!*

- **Given**: Design Graph



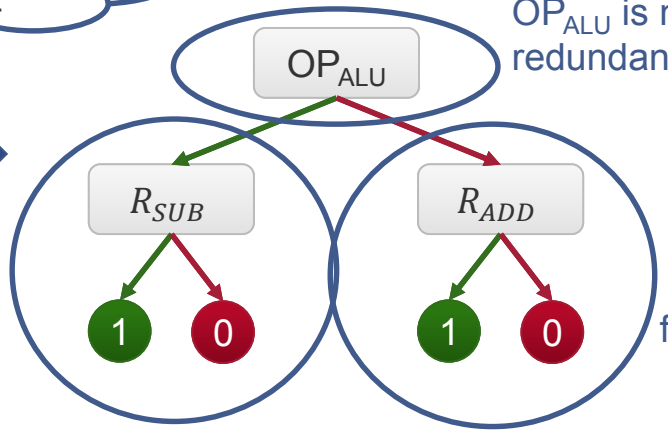


Heuristic: Sort by highest fan-out

Transfer Function of R_{ALU}

$$\neg OP_{ALU} \wedge R_{ADD} \vee OP_{ALU} \wedge R_{SUB}$$

Shannon Expansion



OP_{ALU} is never redundant!

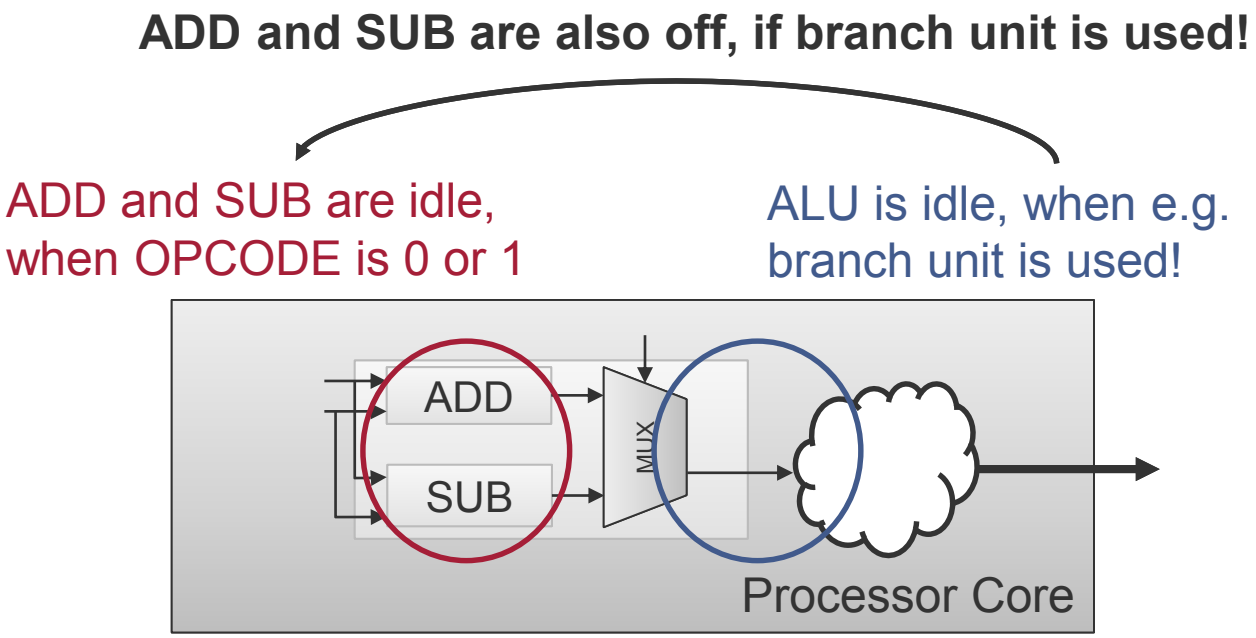
R_{ADD} is redundant for the result of R_{ALU} when OP_{ALU}=1!

R_{SUB} is redundant for the result of R_{ALU} when OP_{ALU}=0!

A Priori Rule Generation

How to derive port-level redundancy rules?

- For each transfer function, we derived a port-level redundancy
- **Issue:** Hierarchy- or dependency-induced redundancy!

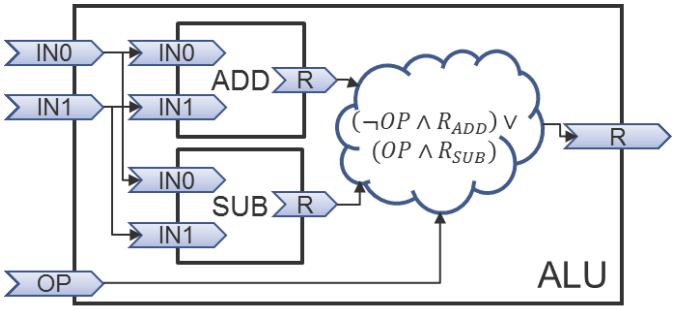


- We solve this issue with a **dependency graph**!

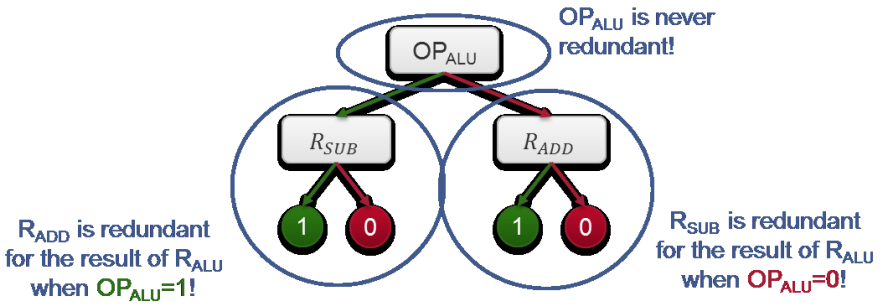
A Priori Rule Generation
Dependency Graph

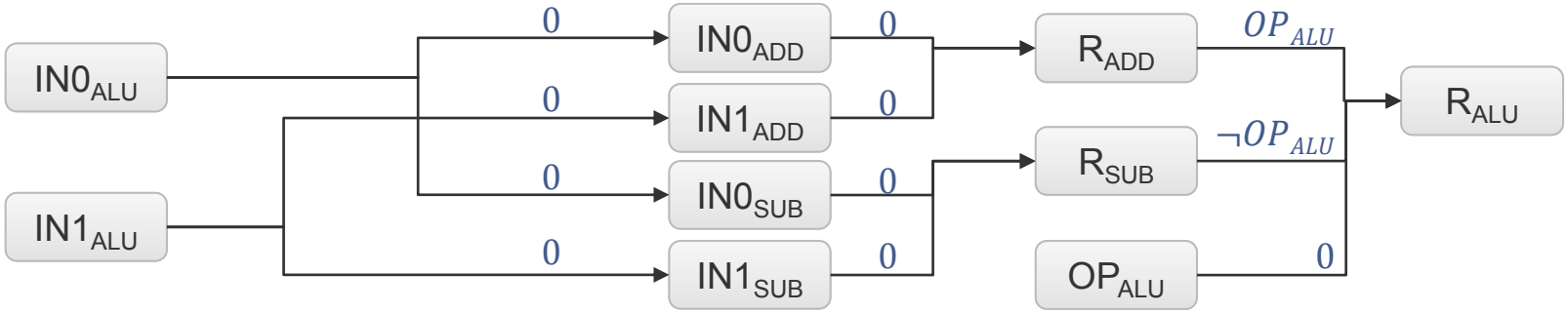
- **Dependency graph** stores interdependencies between ports.
  - *„The value of port X depends on the values of ports A, B,…."*
  - Also stores **port-level redundancy rules**!

- **Given:** Design Graph



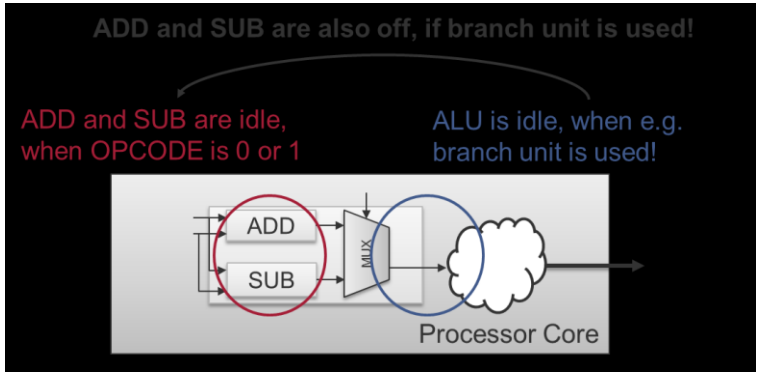
Given: Redundancy BDD for R_{ADD} and R_{SUB}



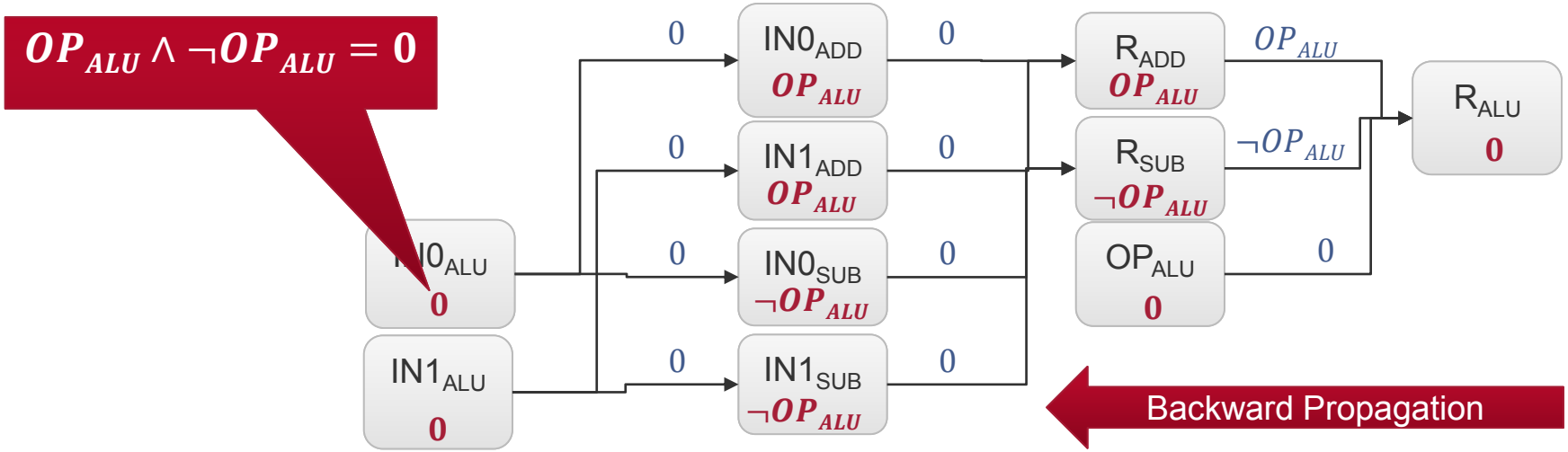


A Priori Rule Generation
Backward Propagation

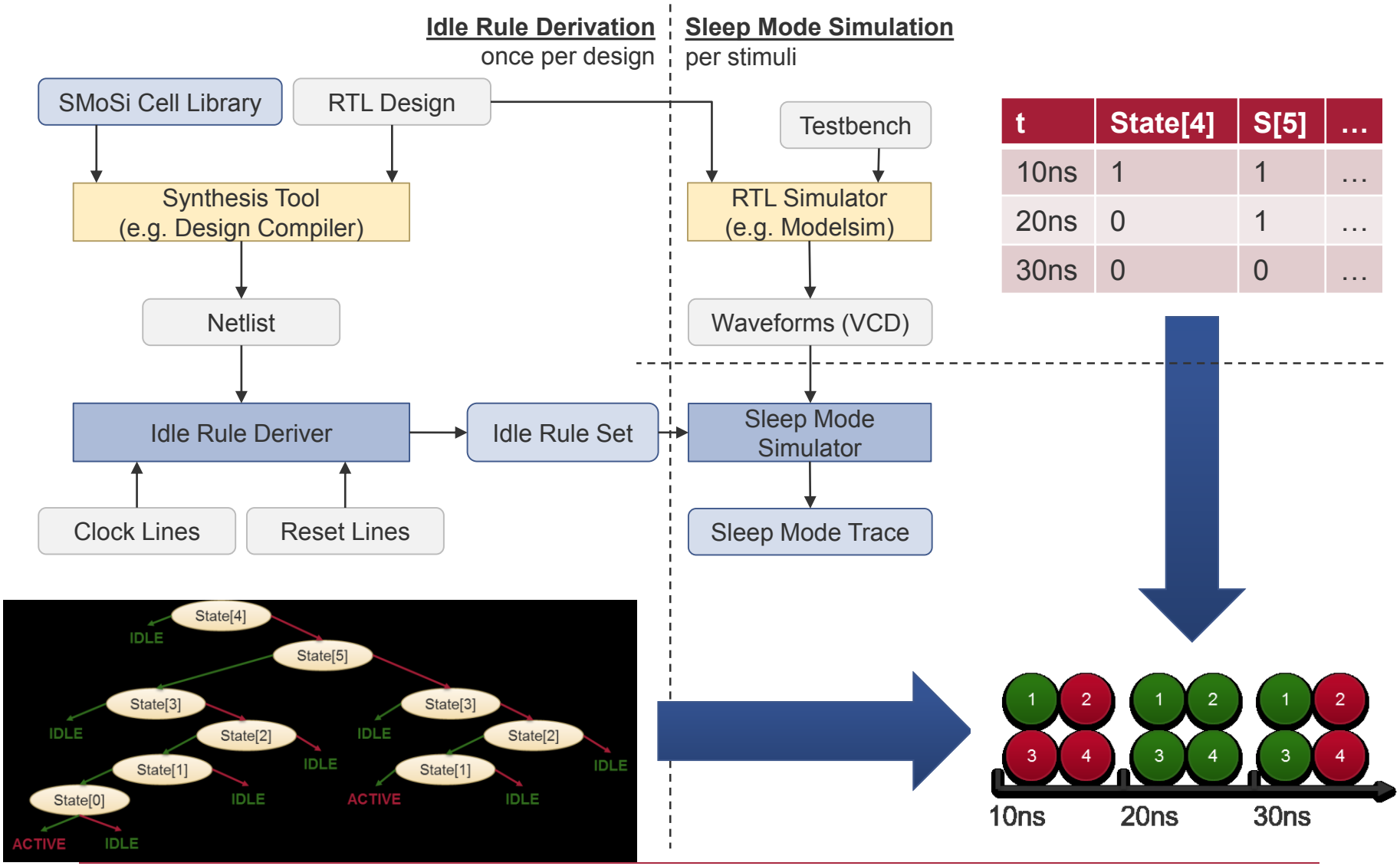
- **Remember:** Hierarchy- or dependency-induced redundancy!



- **Therefore:** Propagate all **local** rules from the edges back to each source node in the dependency graph to get **global** rules!



SMoSi: Sleep Mode Simulation

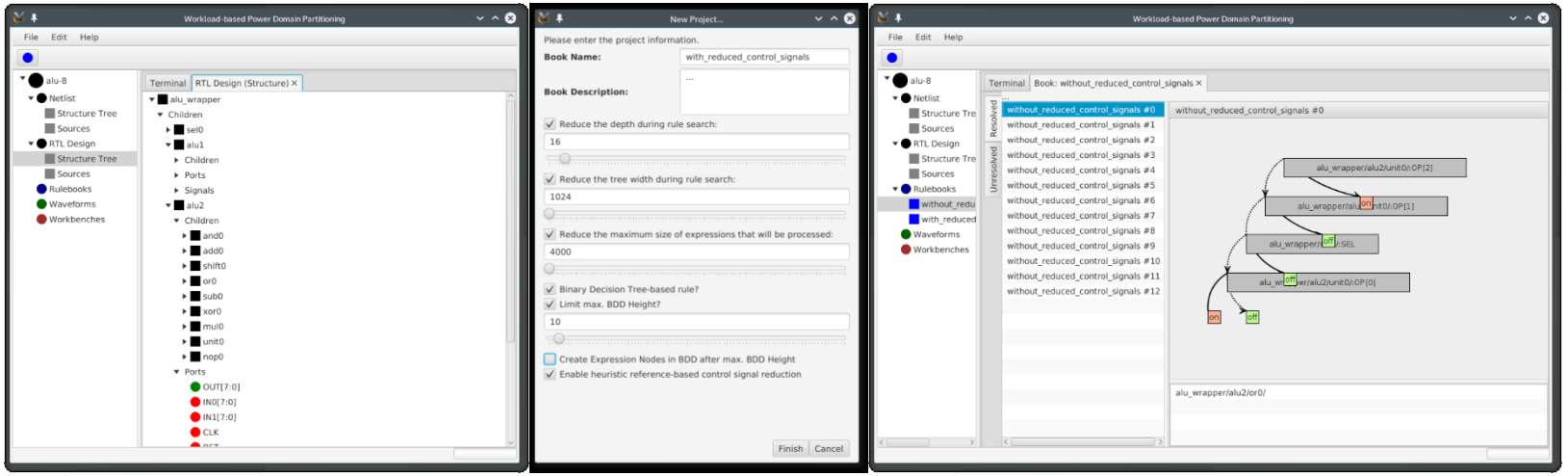


SMoSi under test with open-source designs

2 Experimental Results

Implementation

- Prototypical implementation in Scala (and partially Java)

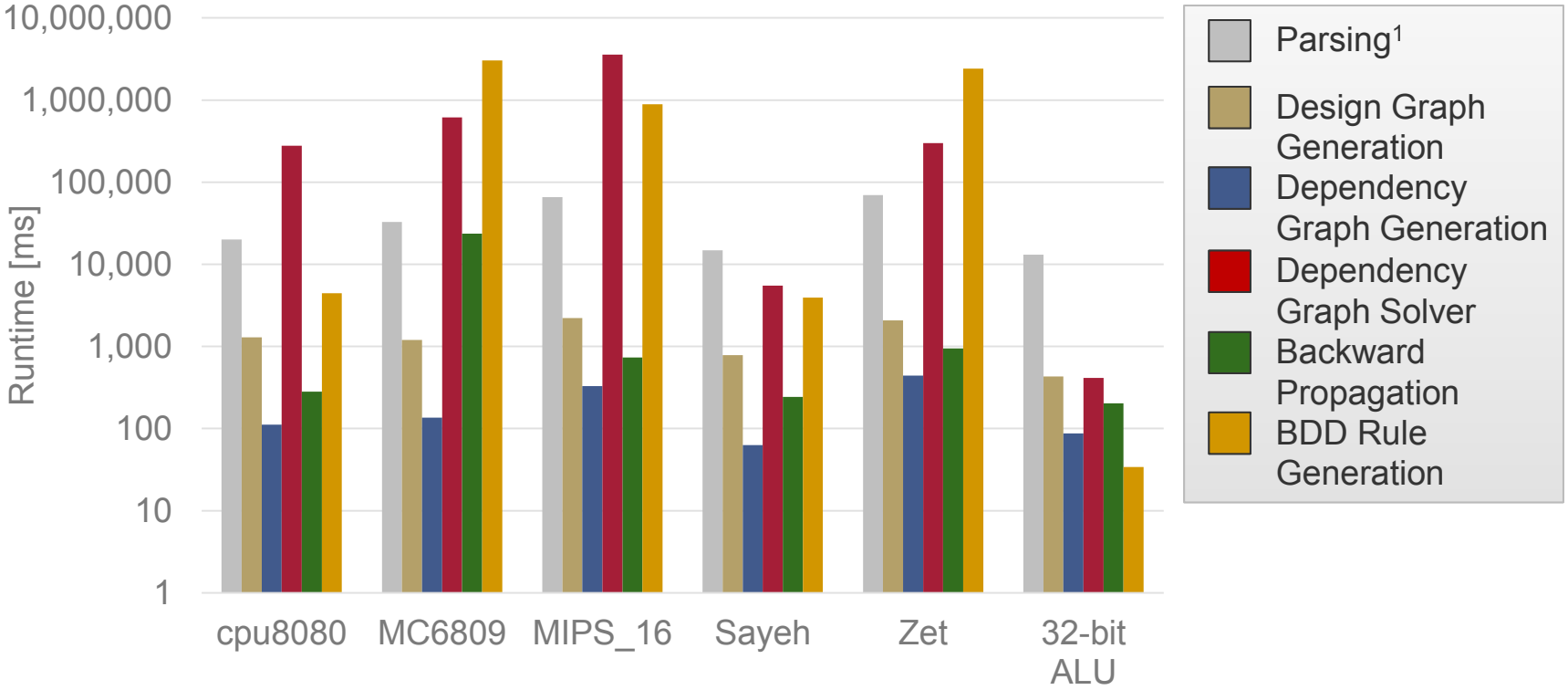


- Pre-built rules and dependency information for **DesignWare components** (e.g. DW03_mux_any, DW01_mult, …)

- **Validation using several testcases:**
  - Several **ALUs** with different characteristics ✓
  - Detection of **pipeline stalls** in a MIPS16 processor ✓
  - **Component idleness** inside a 8080-compatible CPU core ✓

Experimental Results

- 5 Open-Source Designs[1] and a 32-bit ALU have been tested
- 64-core AMD Opteron 6376 (2.3GHz) platform with 256GB RAM, Scientific Linux 6.5, JDK 1.8.0_25 and Scala 2.11.5



[1] Projects from opencores.org

[2] We use the Verilog Parser from EDAUtils (http://www.edautils.com)

Experimental Results

- **Performance results promising, but not yet perfect!**
  - Processing of ALU in reasonable time
  - All other designs could be processed in reasonable time, but with limiting parameters!
  - **Reasons:** Self-made BDD implementation, Shannon Expansion

Design	obdd_depth	max_tf	idle_bdd_depth
cpu8080[1]	16	-	16
mc6809[1]	14	5000	16
mips_16[1]	18	-	18
sayeh[1]	18	5000	18
zet[1]	12	5000	12

[1] Projects from opencores.org

Applications and advancement of SMoSi

3 Conclusion and Future Work

Conclusion

- **State of the Art:** There is no approach to automatically generating sleep mode traces for a design from RTL simulation traces!

- We developed a methodology for the derivation of sleep mode traces at RTL and implemented this methodology in Scala.

- The implementation has been tested successfully for small and middle sized designs!

- **Future Work:**
  - Integration of an existing BDD library for performance gain
  - Evaluation of alternative diagram types (KFDDs, ZDDs, …) and heuristics for identifying control signals
  - Coupling with existing design partitioning approaches

Any questions?

Dustin Peterson

Eberhard Karls Universität Tübingen
Lehrstuhl für Eingebettete Systeme
Fon: +49 7071 - 29 – 75458
Fax: +49 7071 - 29 – 5062

E-Mail: dustin.peterson@uni-tuebingen.de