Architecture and Design Methodology for Autonomic Systems-on-Chip (ASoC)

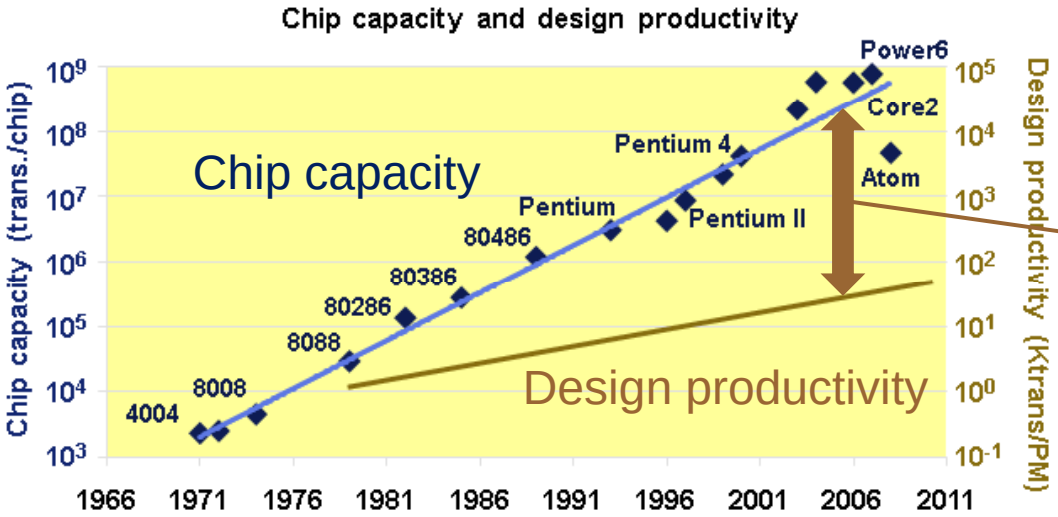
Andreas Bernauer[1], Abdelmajid Bouajila[2], Oliver Bringmann[3],
Dirk Fritz[1], Johannes Zeppenfeld[2], Walter Stechele[2],
Andreas Herkersdorf[2], Wolfgang Rosenstiel[1,3]

[1]Universität Tübingen

[2]Technische Universität München

[3]FZI - Forschungszentrum Informatik

Major challenges of future SoCs

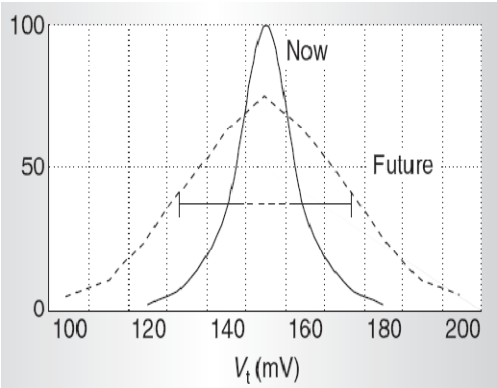


Design gap

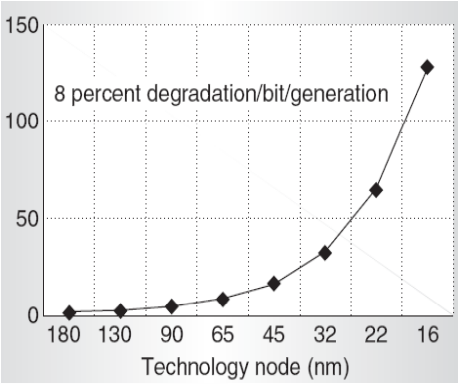
Source: ITRS 2008

Reliability issues

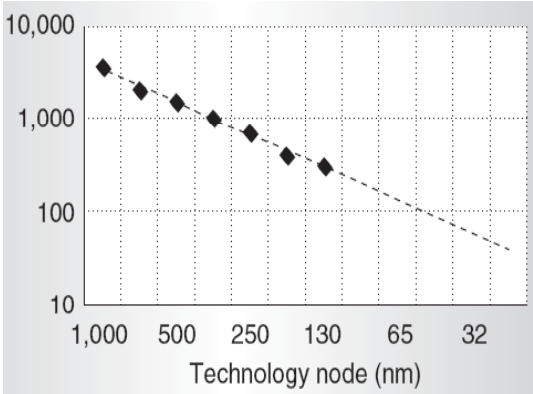
Device Variations (Target Voltage)



Soft Errors (Relative Failure Rate)

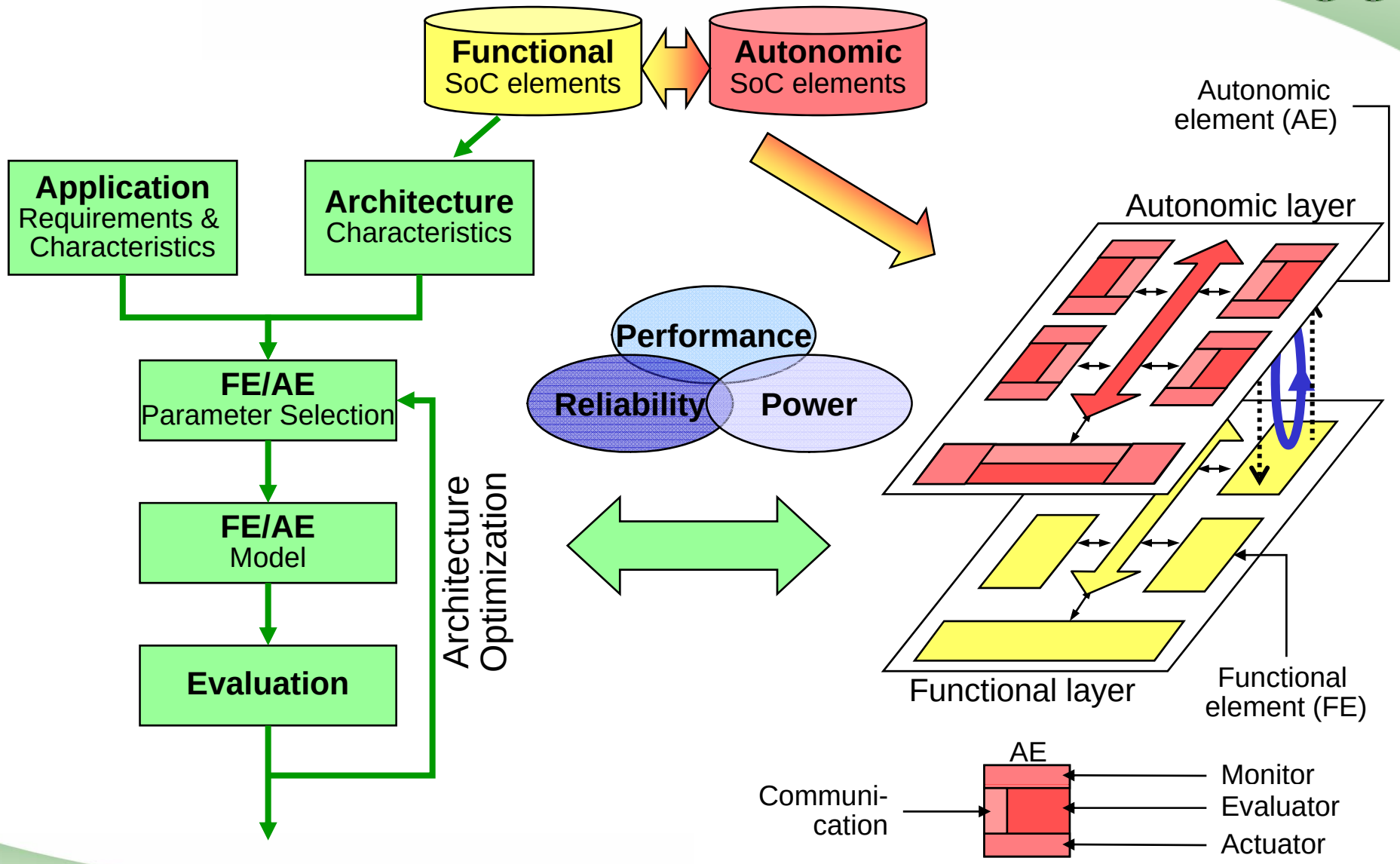


Dopant atoms in the transistor channel

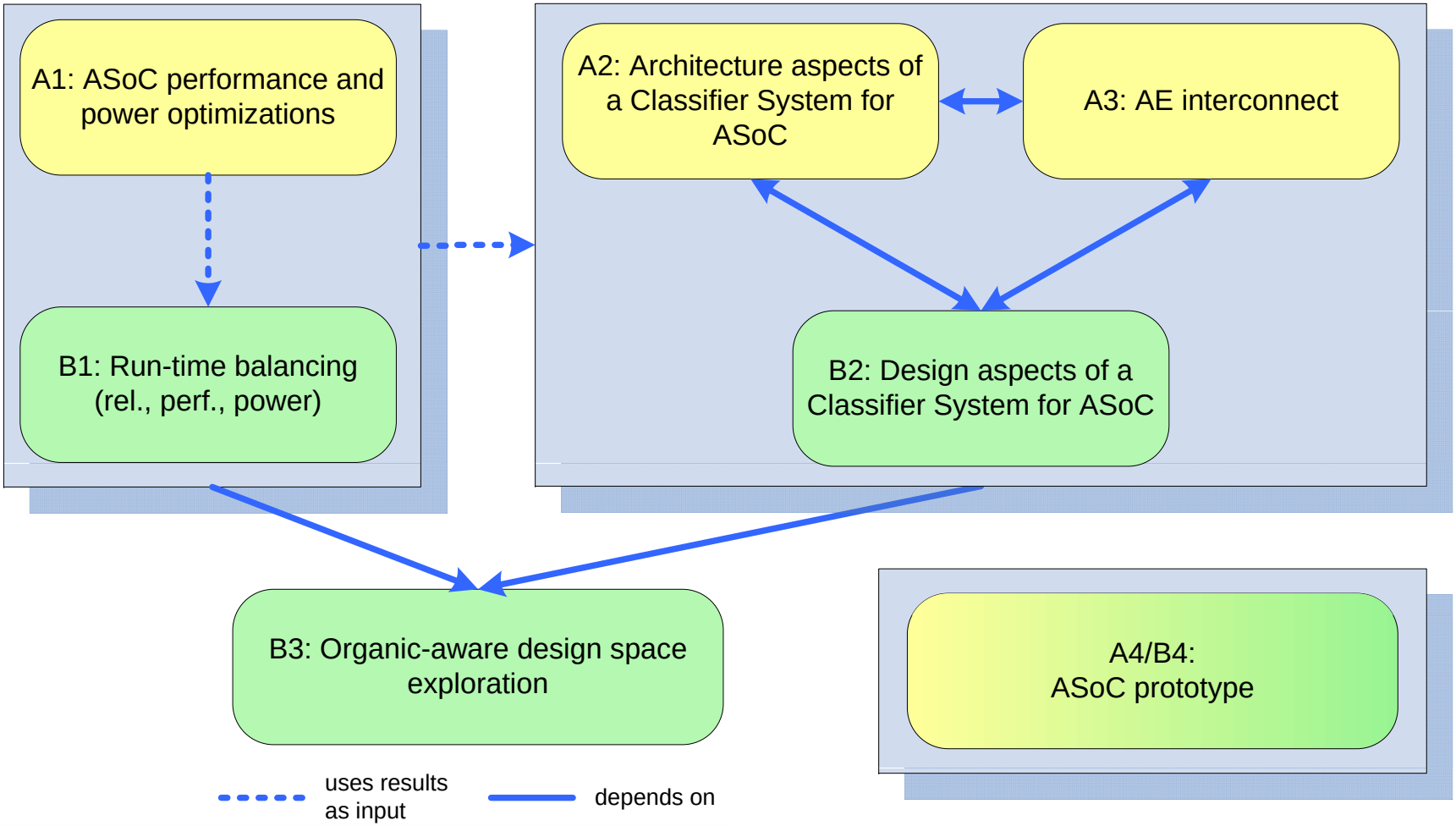


[Borkar06]

Project reminder

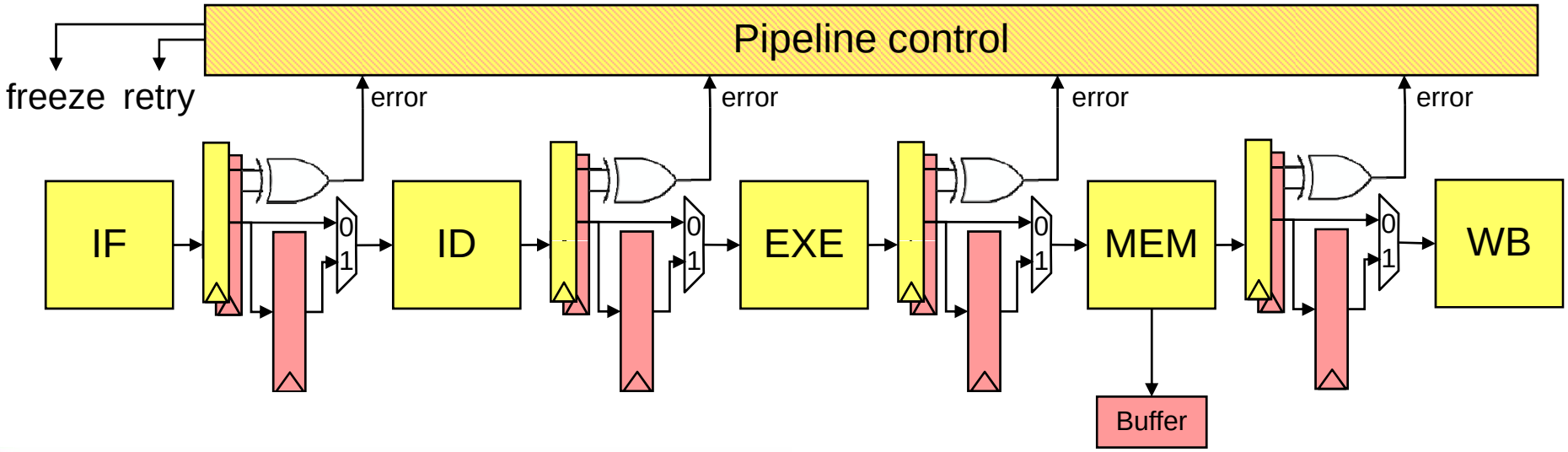


Phase 2 Work Packages



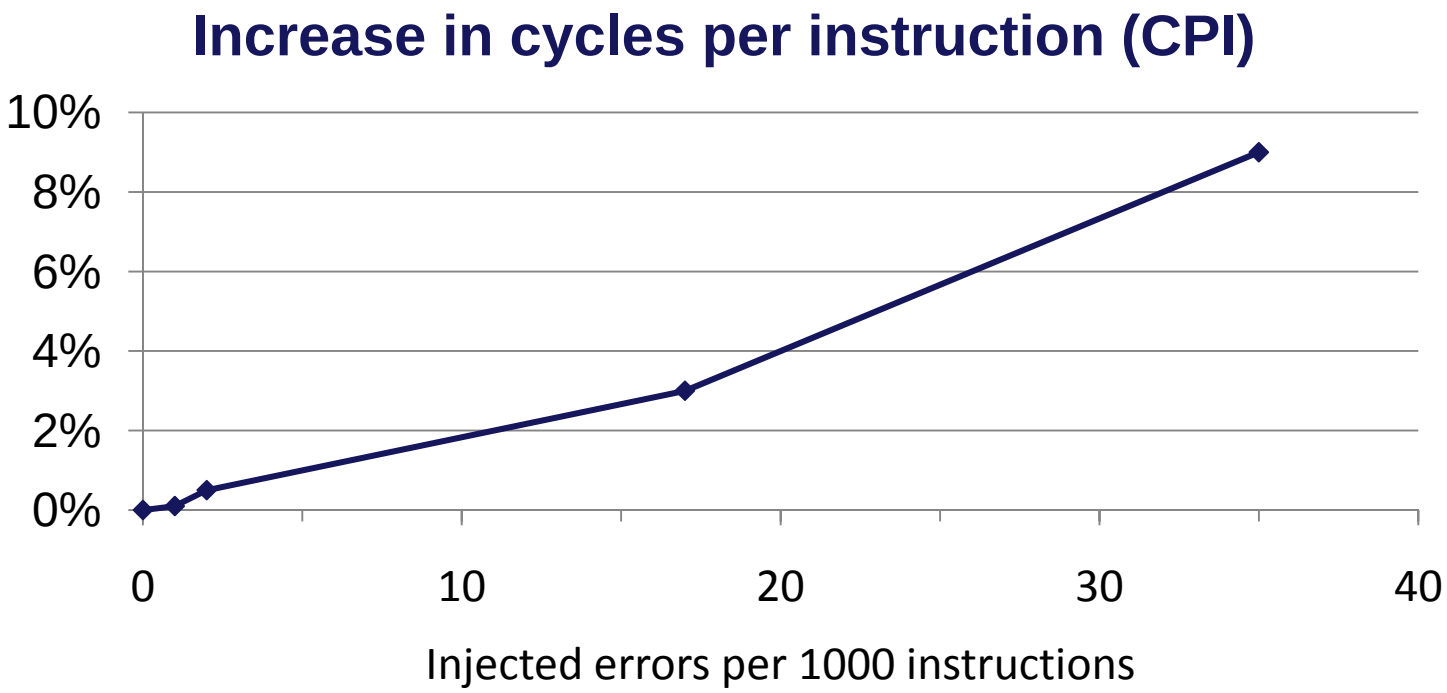
A1: Self-healing CPU Pipeline [VLSI06]

- Error detection using Nicolaidis shadow registers
- History registers keep track of latest pipeline stage registers
- No pipeline flushing necessary → fixed 2-cycle penalty
- Implemented in Leon3 processor



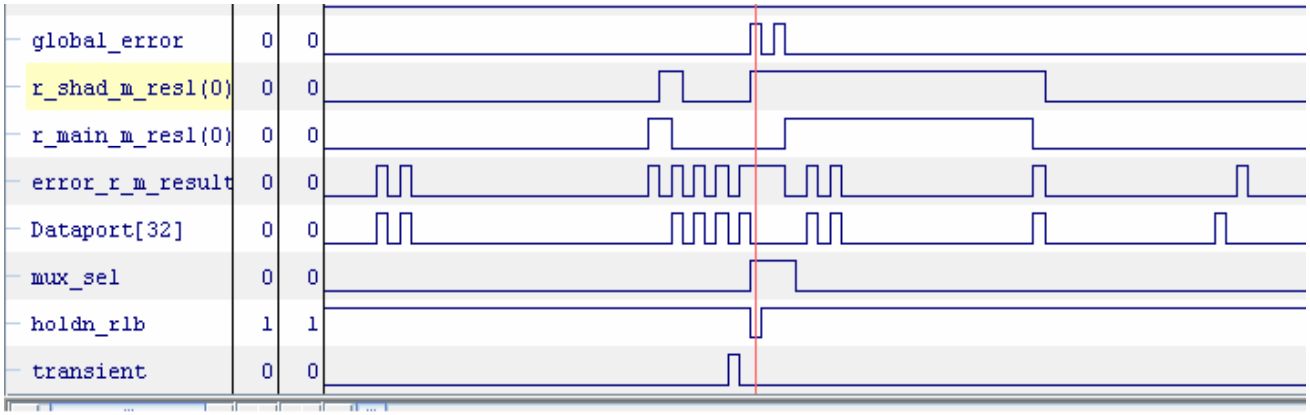
A1: Simulation results

- Leon3 Modelsim simulations
- Error injection and detection using Mibench testbench



A4: FPGA prototyping

- Self-healing Leon3 pipeline running on a Virtex4 board
  - Error injection in FPGA
  - Error detection and correction
  - Will be a building block for ASoC MPSoC FPGA demonstrator

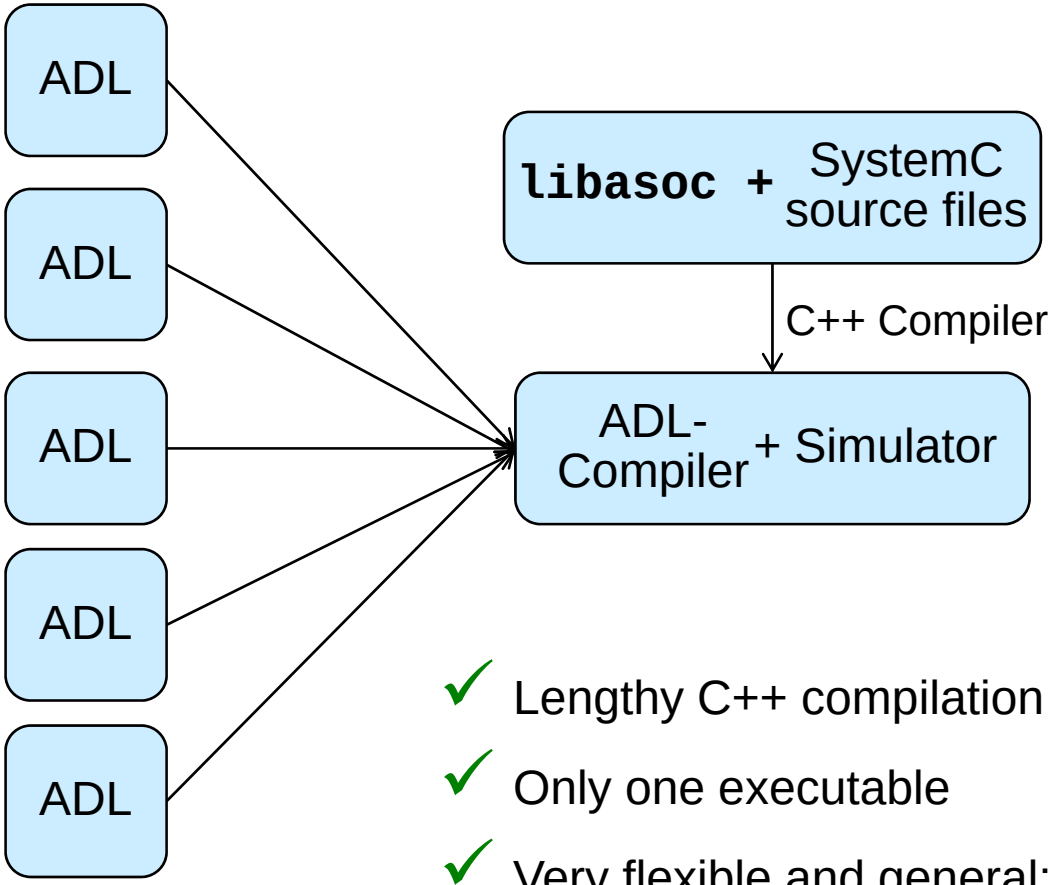


ChipScope measurements

A2/B3: Architecture description language for SystemC

- Design space exploration requires architecture description language (ADL) for SystemC
- Most existing ADL are special purpose.
- Few ADL address SystemC (LISA, ArchC), however focus on instruction set architectures.
- No ADL is flexible enough to describe new ASoC architecture.

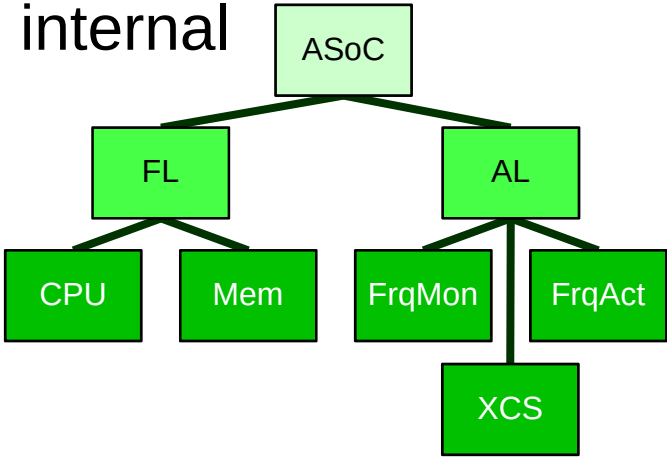
`libasoc` – an ADL for SystemC

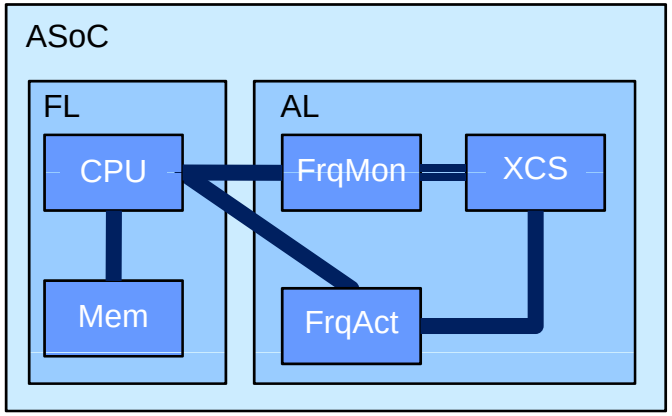


- ✓ Lengthy C++ compilation happens only once.
- ✓ Only one executable
- ✓ Very flexible and general: full SystemC

`libasoc` – internals

- At elaboration, read in ADL and create internal representation.
- The internal representation contains
  - Hierarchy information
  - Dependencies
  - Port bindings
  - Module parameters
- Construct SystemC hierarchy.
- Start simulation.





libasoc – code examples

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ASoCsim name="ASoC" type="Simulator">

  <Module name="FL" type="Layer">
    <Module name="CPU" type="FE">
      <param name="frequency" value="2000MHz"
             type="frequency"/>
      <port name="mem" target=".MEM.bus"/>
    </Module>
  </Module>

  <Module name="AL" type="Layer">
    <Module name="FrqMon" type="monFreq">
      <depend target=":.ASoC.FL.CPU"/>
    </Module>
  </Module>

</ASoCsim>
```

Architecture description

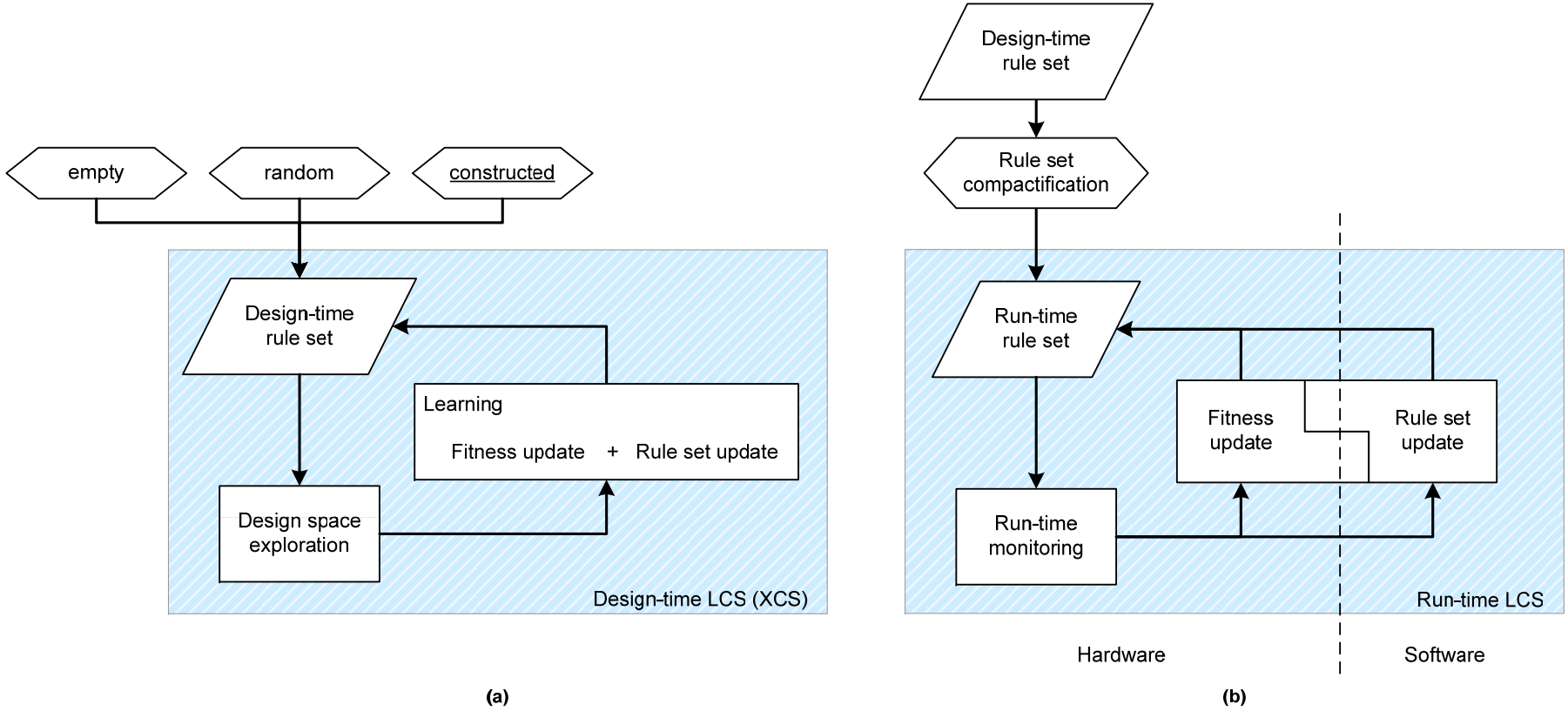
libasoc – code examples

```
FE::FE(module_name& mn, module_desc& md)
: Module(mn, md)
{
    this->freq = md.param["frequency"].getValue<frequency>();
}


monFreq::monFreq(module_name& mn, module_desc& md)
: Module(mn, md)
{
    deferReference(md, this->targetFE, md.depend.get());
}
```

Behavioral description in SystemC

Design-time and run-time learning

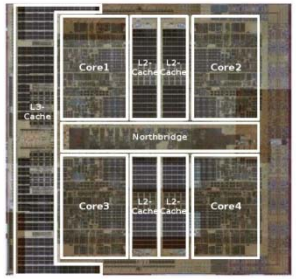


Design-time learning (XCS)

Run-time learning (LCT)

B2: Evaluation of XCS for SoC control [Bernauer08a]

- Setup: simulate workloads on AMD Opteron QuadCore
  - LR decomposition, folding, matrix multiplication
- Estimated parameters:
  - Static Power $P_s = V_{DD} \cdot N \cdot k_{\text{design}} \cdot \hat{I}_{\text{leak}}$
  - Dynamic Power $P_d = \alpha \cdot C_L \cdot V_{DD}^2 \cdot f_p$
  - Temperature (HotSpot) [Huang04]
  - Soft error rate [Zhu06] $\lambda(f) = \lambda_0 \cdot 10^{\frac{(1-f)\cdot d}{1-f_{\min}}}$
  - Timing errors [Gold03] $t_{\text{av}}(T) = t_{\text{av}}^0 + t_{\text{delay}}(T)$
- 2 steps: XCS learning & XCS acting
- Goal: find optimal frequency & voltage



Step 1: XCS learning

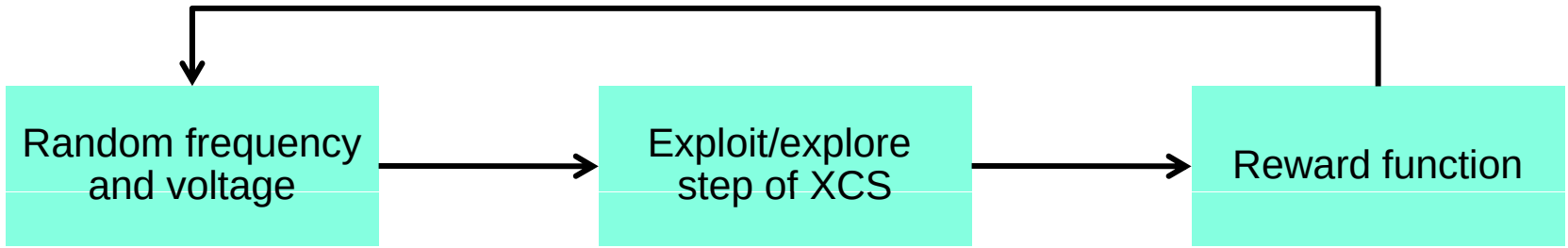
- Reward function: performance, power, reliability

$$R(f,p,t,v) = w_1 \cdot \boxed{\frac{f}{f_{\max}}} + w_2 \cdot \boxed{(1 - \frac{p}{p_{\max}})} + w_3 \cdot \boxed{\mathrm{rel}(t,v,f)}$$

$$\text{with } \mathrm{rel}(t,v,f) = \begin{cases} 0 & \text{if error} \\ 1 & \text{otherwise} \end{cases}$$

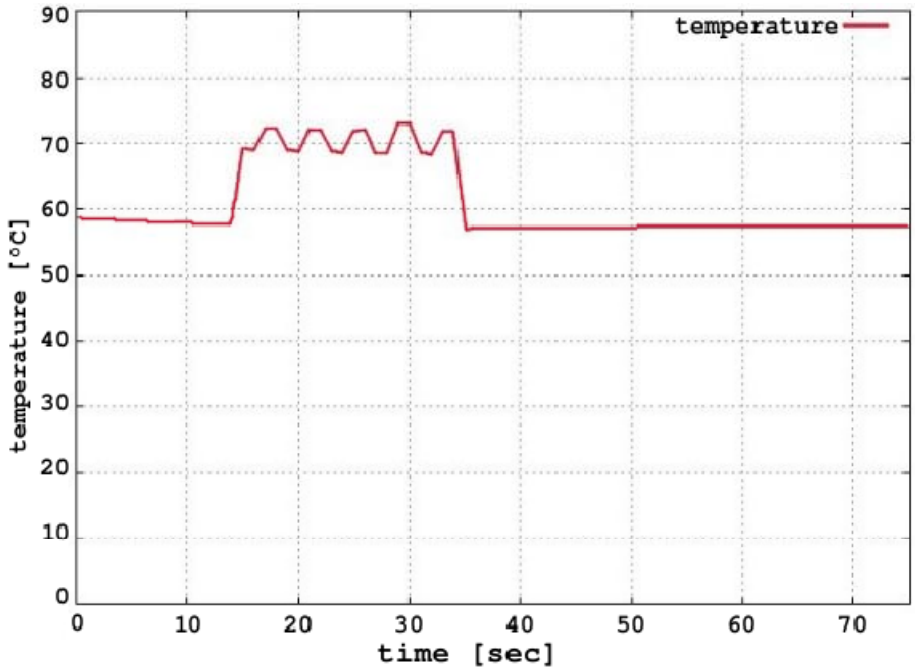
$$\text{here}: w_1 = 200, w_2 = 35, w_3 = 200$$

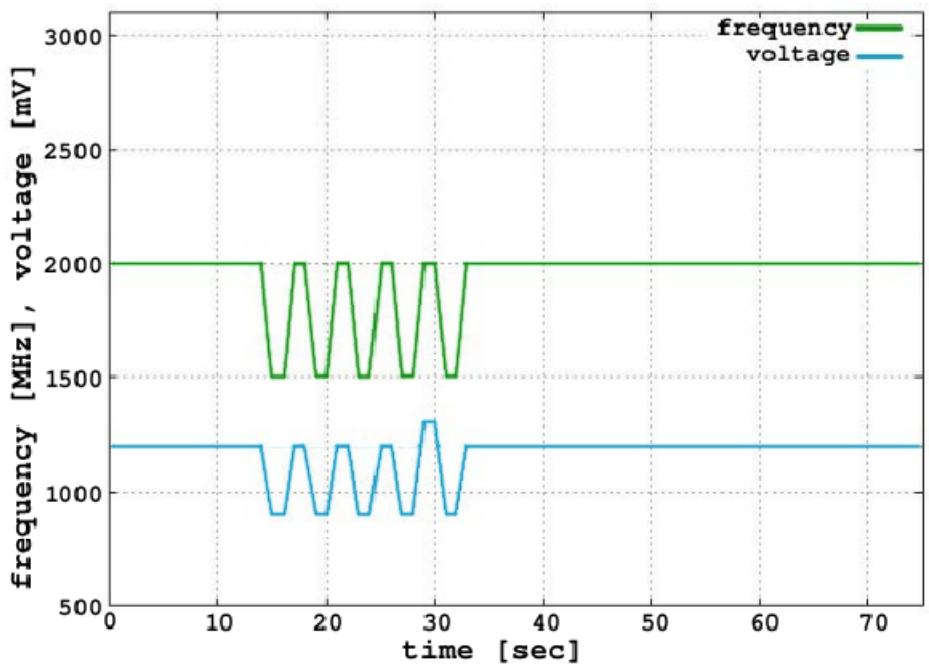
- Learning cycle



Step 2: XCS acting

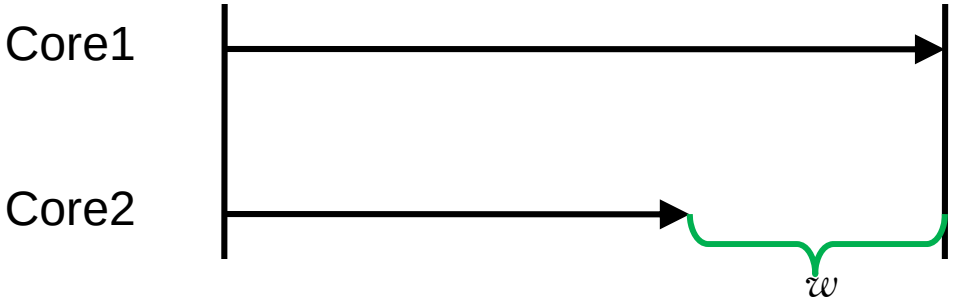
- Under normal conditions, XCS finds optimal operating point.
- With increased ambient temperature (+15 K):
  - XCS finds optimal operating point.
  - However, system oscillates.





Learning without genetic algorithm

- Optimize dual-core application:

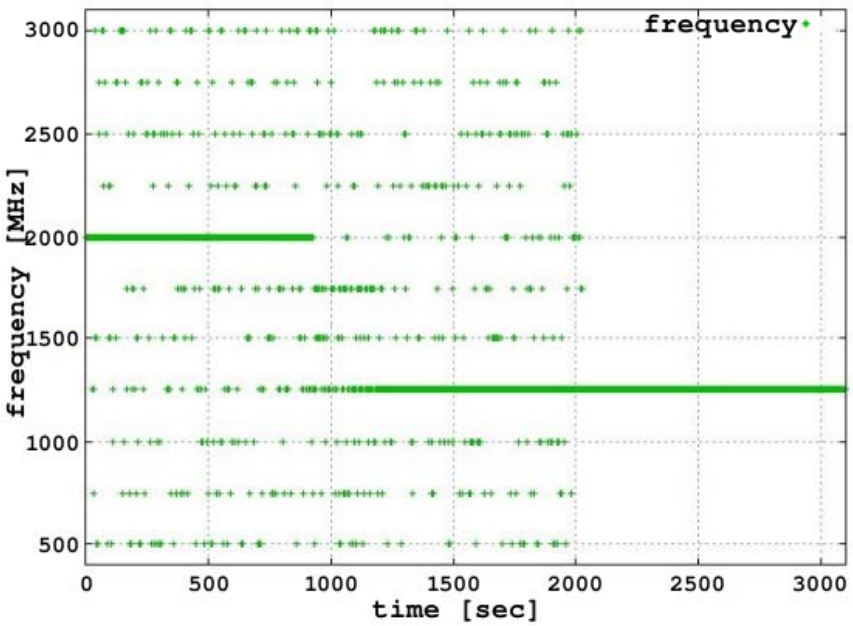


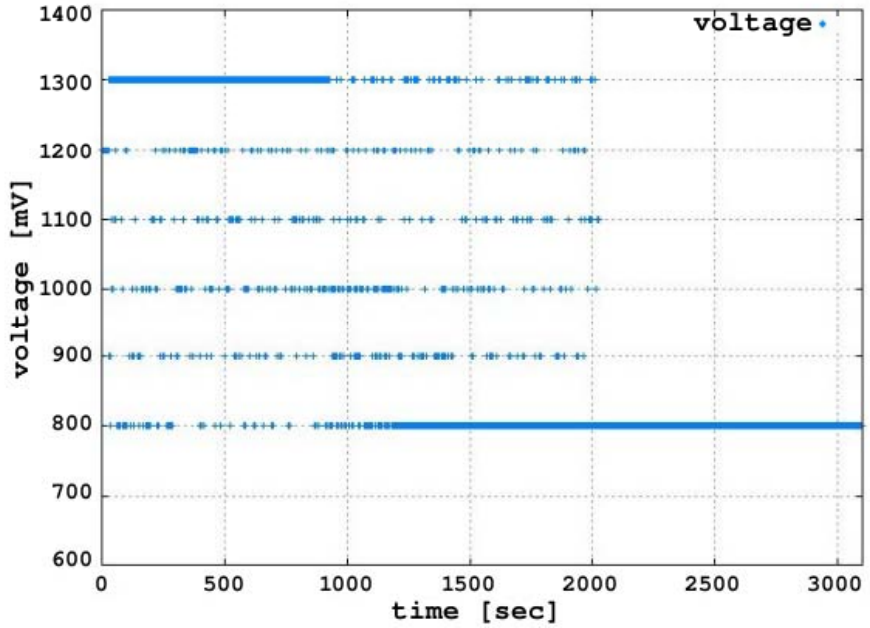
- Reward function to minimize waiting time:

$$R(f,p,t,v,w) = w_1 \mathrm{time}(w) + w_2 \left(1 - \frac{p}{p_{\max}}\right) + w_3 rel(t,v,f)$$

$$\mathrm{time}(w) = \begin{cases} 1 - \frac{w}{w_{\max}} & \text{if the waiting time of Core 1=0} \\ 0 & \text{otherwise} \end{cases}$$

Learning without genetic algorithm



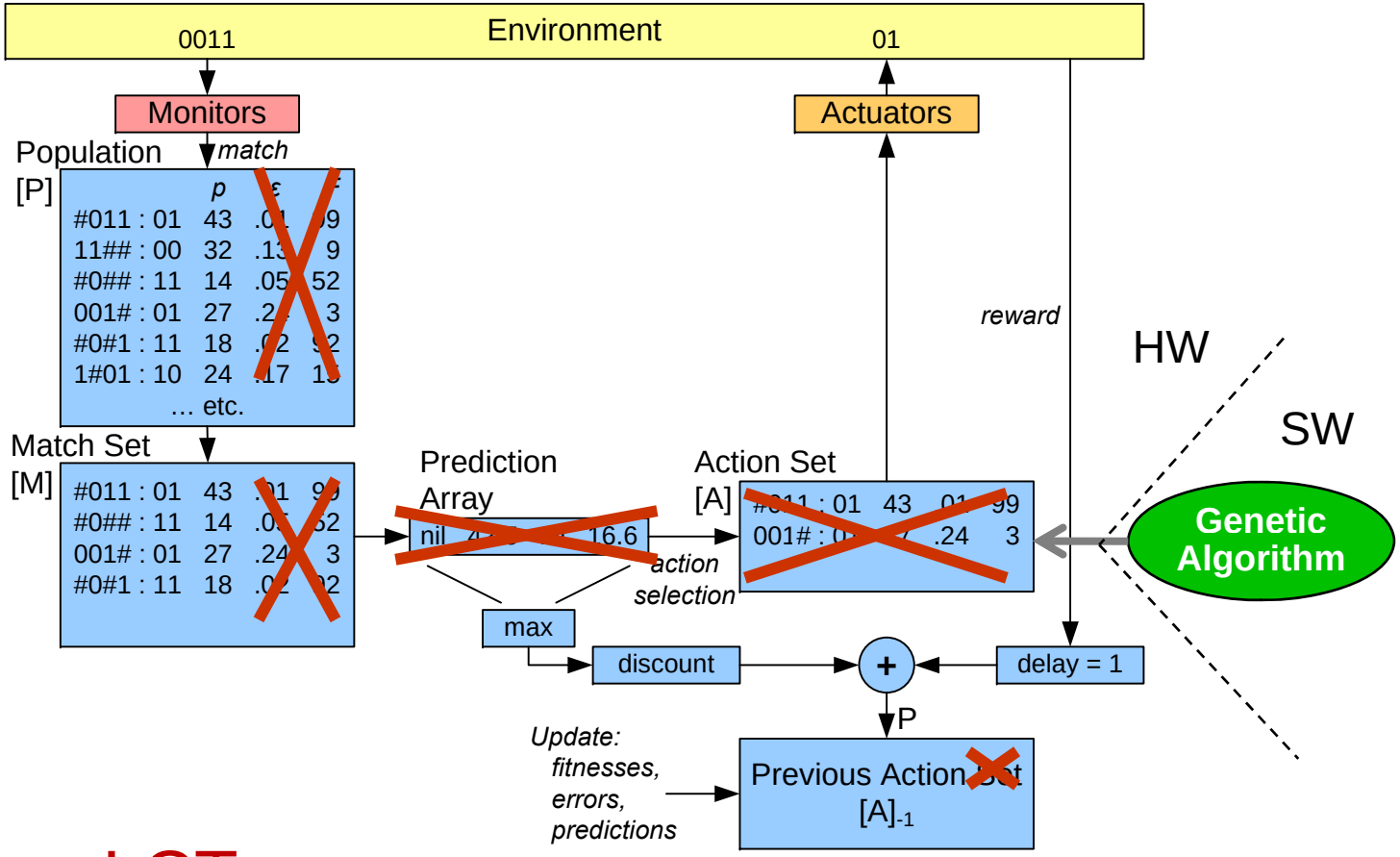


- Yes, XCS can learn without genetic algorithm.
- Long delay due to 2s time constant to measure temperature.

Summary XCS Evaluation

- XCS can find optimal operating points in CPUs.
- XCS tolerates changing environments.
- XCS tolerates changing reward functions.
- XCS can learn without genetic algorithm.
- First step towards the goal that the designer does not have to specify everything in detail.

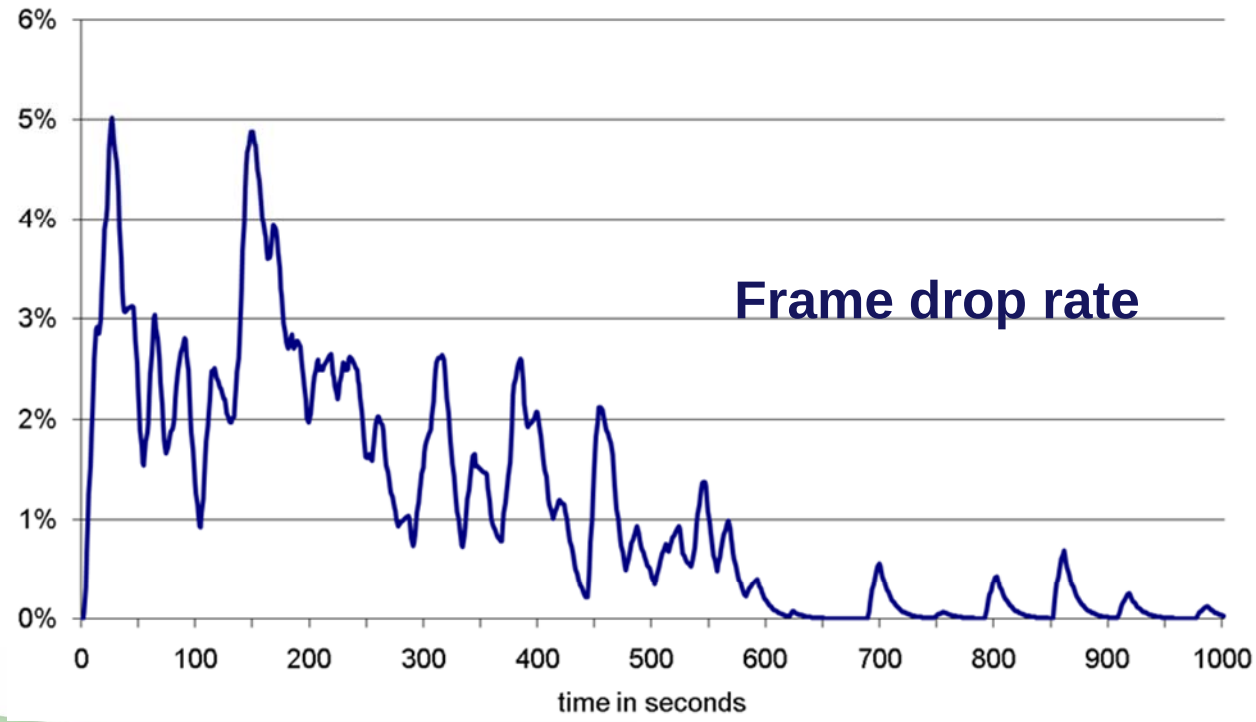
A2/B2: XCS and LCT



LCT

LCT learning results

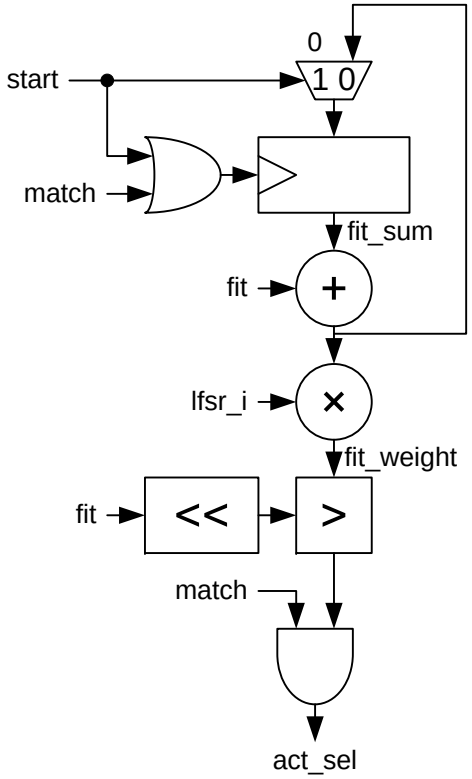
- Simulation: dual-core video application where one core fails sometimes.
- Goal: learn to adjust the frequency of non-failing core quickly.



A2: LCT implementation [Zeppenfeld08]

Preliminary Synthesis Results
(Xilinx Virtex-II Pro FPGA, without fitness update)

HW LCT	Leon 2	%
38 Slices	2979	1.3
45 Flip-flops / 58 LUTs	1591 / 5450	< 3
1 BRAM, 1 Multiplier	6 BRAMs, 0 Mults	17
Period: 7.6 ns	Period: 23.5 ns	-
Frequency: 131 MHz	Frequency: 43 MHz	-

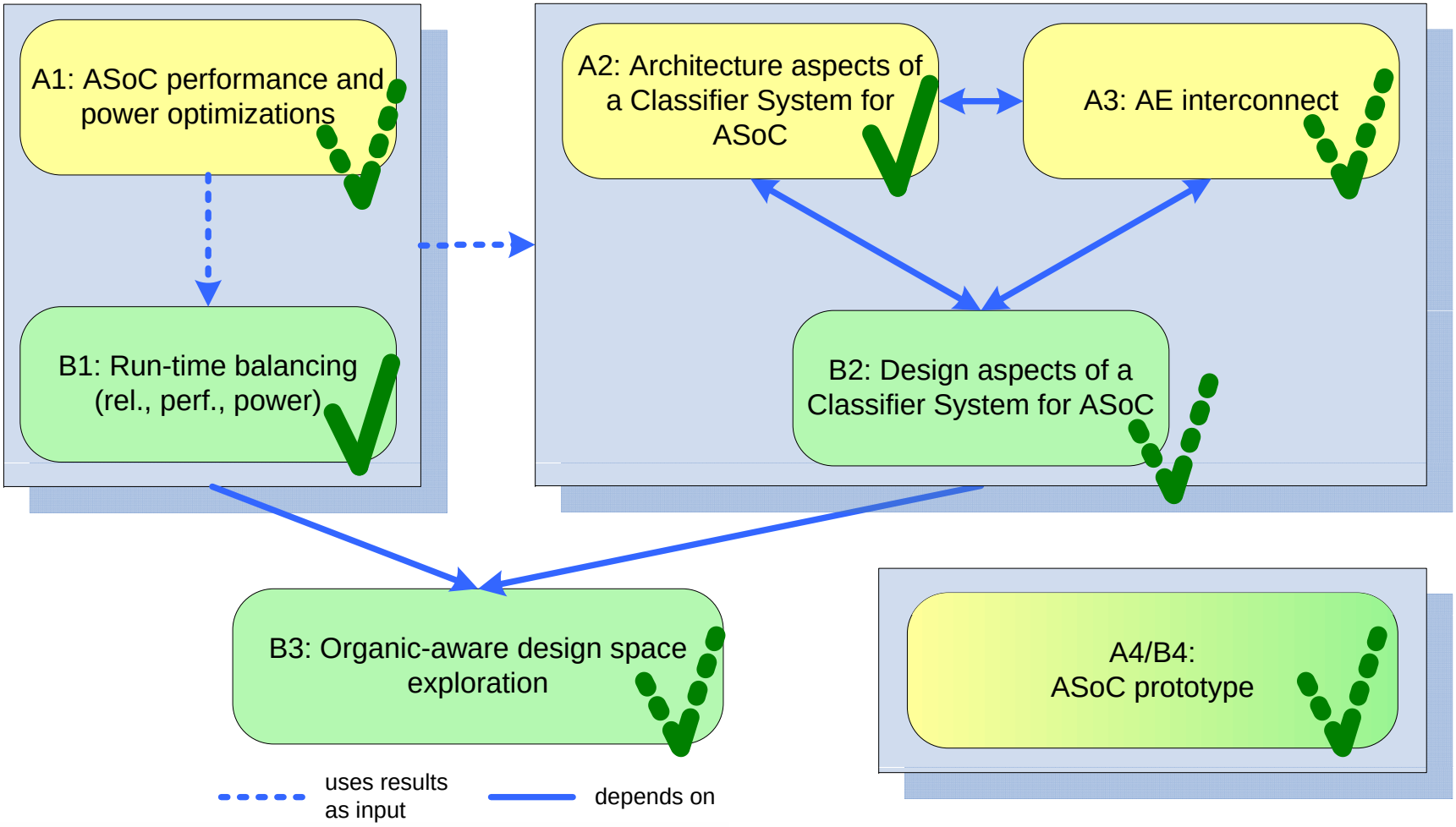


Weighted reservoir sampling
for action selection

Cooperations

- Team of Prof. Reif
  - Verification of self-x properties based on logic model
  - Tools for reliability estimations
- Team of Prof. Fey
  - Increase reliability of marching pixels.
- Teams of Prof. Müller-Schloer/ Prof. Schmeck
  - LCS concept and implementation
- Team of Prof. Maehle/ Prof. Brockmann
  - Cooperation on HW/SW Learning
- Team of Prof. Ernst
  - Interested in reliability estimation.
- 2nd LCS workshop at Karlsruhe, June 19/20, 2008
  - Well visited with 25 participants
- Organic Computing Workshop at annual GI conference, September 12, 2008

Summary


A1: ASoC performance and power optimizations
B1: Run-time balancing (rel., perf., power)
A2: Architecture aspects of a Classifier System for ASoC
A3: AE interconnect
B2: Design aspects of a Classifier System for ASoC
B3: Organic-aware design space exploration
A4/B4: ASoC prototype
uses results as input
depends on

Publications and References

- [Bernauer08a] A Bernauer, D Fritz, W Rosenstiel, *Evaluation of the Learning Classifier System XCS for SoC run-time control ,* Lecture Notes in Informatics, Vol. 134, p.761-768, Springer, Gesellschaft für Informatik
- [Bernauer08b] A Bernauer, D Fritz, B Sander, O Bringmann, W Rosenstiel, *Current state of ASoC design methodology*, http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=1564, ISSN 1862-4405
- [Herkersdorf08] A Herkersdorf, J Zeppenfeld, A Bouajila, W Stechele, *Hardware-Supported Learning Classifier Tables in Autonomic Systems on Chip*, Dagstuhl Seminar 08141, March 30 - April 4, 2008
- [Lankes07] A Lankes, T Wild, J Zeppenfeld, *Power estimation of Variant SoCs with TAPES.* In: Euromicro-DSD 2007.
- [Zeppenfeld08] J Zeppenfeld, A Bouajila, W Stechele, A Herkersdorf, *Learning Classifier Tables for Autonomic Systems on Chip,* Lecture Notes in Informatics, Vol. 134, p.769-776, Springer, Gesellschaft für Informatik
- [Gold03] A Gold, A Kos, *Temperature Influence on Power Consumption and Time Dealy*, Dep. Electronics, Cracow, Poland, 2003.
- [Huang04] W Huang, MR Stan, K Skadron, K Sankaranarayanan, S Ghosh, S Velusamy, *Compact Thermal Modeling for Temperature-Aware Design,* DAC 04.
- [Zhu06] D Zhu, *Reliability-Aware Dynamic Energy Management in Dependable Embedded Real-Time Systems,* RTAS'06, IEEE Computer Society, 2006, p. 397-407