Implementation of Self-Healing Asynchronous Circuits at the Example of a Video-Processing Algorithm

T. Panhofer W. Friesenbichler **A. Steininger**
Vienna University of Technology

Outline

- Motivation & Objective
- Asynchronous Logic
- Self-Healing Concept
- Case Study: SH implementation of video processing algorithm
- Experimental Results (& Lessons Learnt)
- Conclusion & Outlook

The Nanoscale Challenges

- significant parameter variations
  - threshold voltages, delays, leakages,…
- increased rate of transient faults
  - lower voltage, smaller critical charge,…
- increasing danger of permanent faults
  - more functions/chip, higher temperature
- …

Resulting Needs

- significant parameter variations
  - need robust design methods that are inherently able to cope with these variations
- increased rate of transient faults
  - need fault tolerance or robustness
- increasing danger of permanent faults
  - need self-repair or „self-healing“
- …

Why Use Asynchronous Logic?

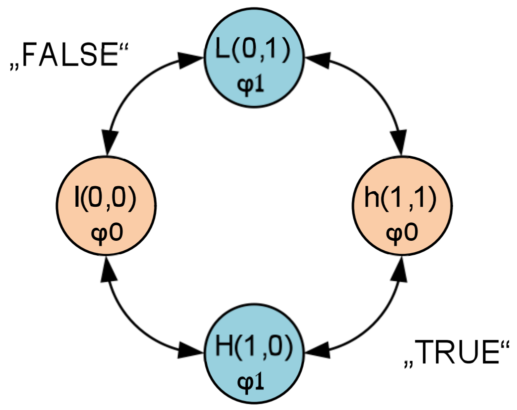
- „delay insensitive“ operation
  - based on local handshaking (closed loop),
  - not on global clock (open loop)

  high robustness in time domain

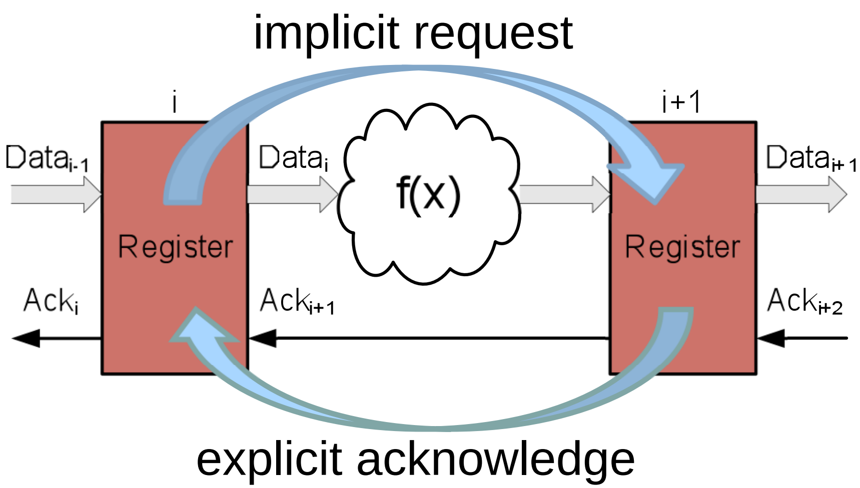
- two-rail coded data

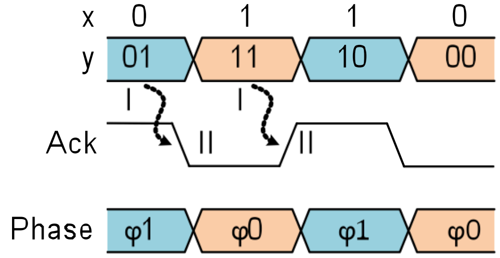
  high robustness in value domain

FSL – How does it work?



- dual-rail encoded data
- two representations for HI/LO
- tokens in alternating „phases“





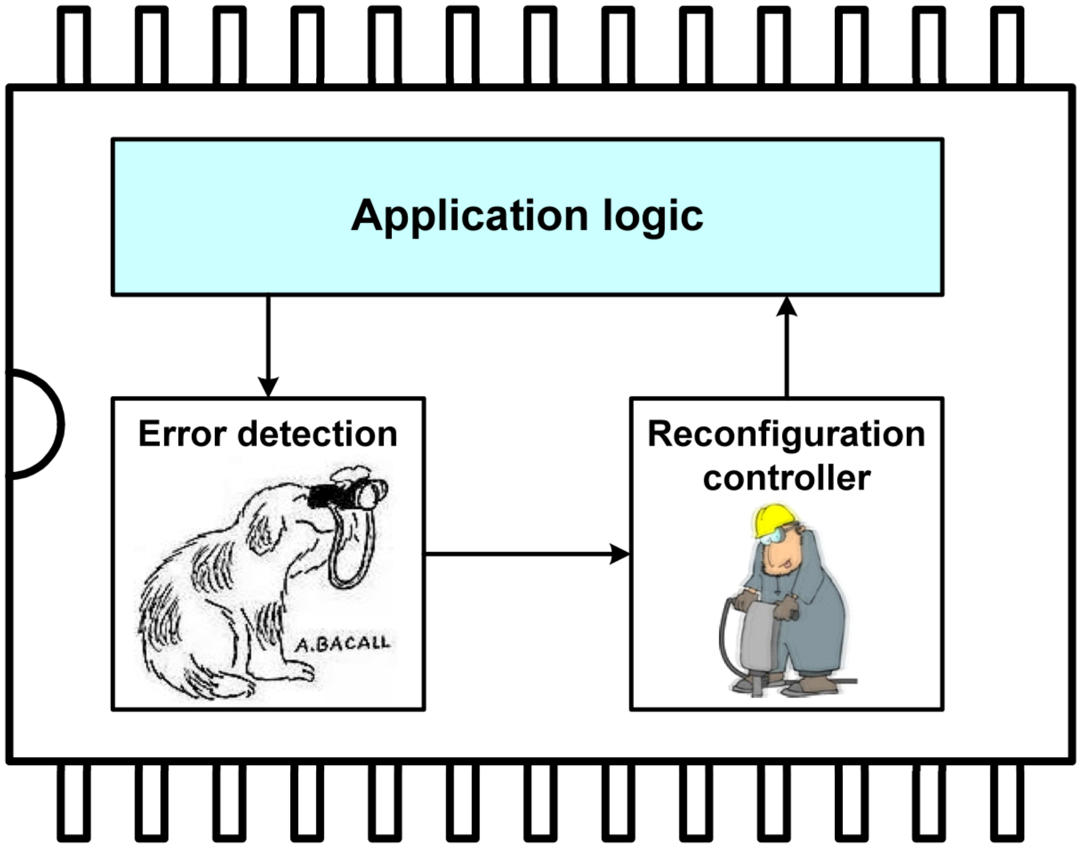
How far does this get us?

- ✓ significant parameter variations
  - delay-insensitive logic has a robust timing that can tolerate (virtually) all variations
- ✓ increased rate of transient faults
  - two-rail coding, robust timing
- increasing danger of permanent faults
  - still need self-repair or „self-healing“

Requirements for „Self-Healing“

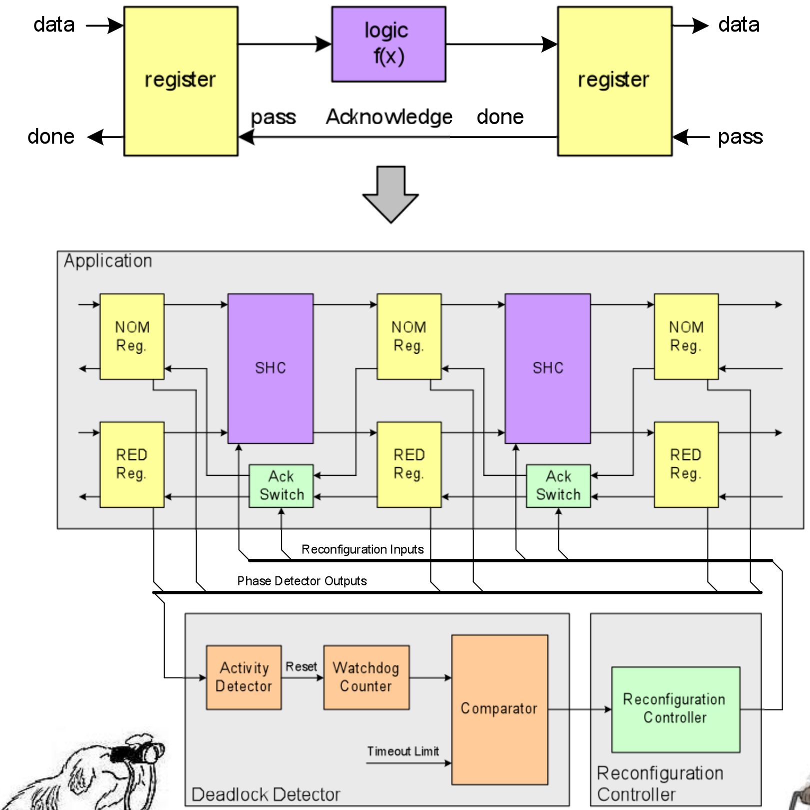
- detection of (permanent) error
  - ☺ DI logic tends to stop working in this case
- identification of faulty cell
  - ☺ handshake signals tend to point there
- fault removal
  - ☺ temporal robustness makes re-routing easier

Self-Healing Concept (1)

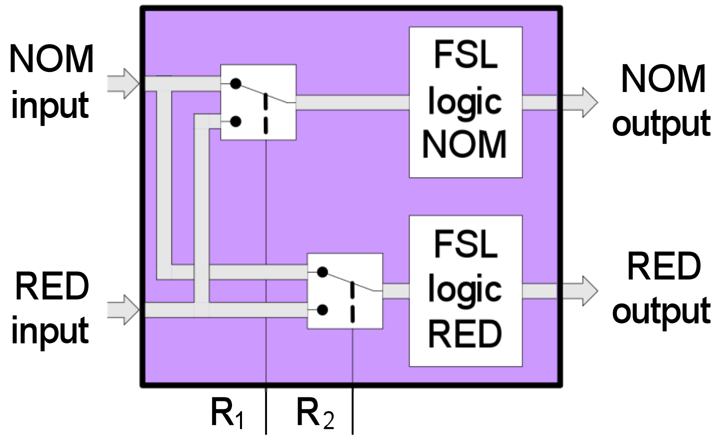


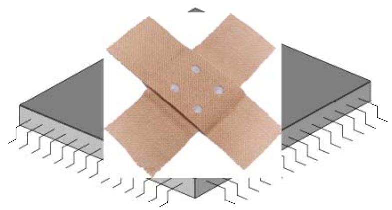
Self-Healing Concept (2)

Transformation



Self-Healing Cell





What‘s the Benefit over TMR?

- both approaches tolerate first fault
  - TMR without interruption of service (2oo3)
  - selfhealing possibly with interruption (1oo2)
- self-healing is more fine-grained
  - more options to bypass defective element
  - no need to rely on „luck“ (next defect not in remaining operative nodes)

Why not use dynamic Reconfig.?

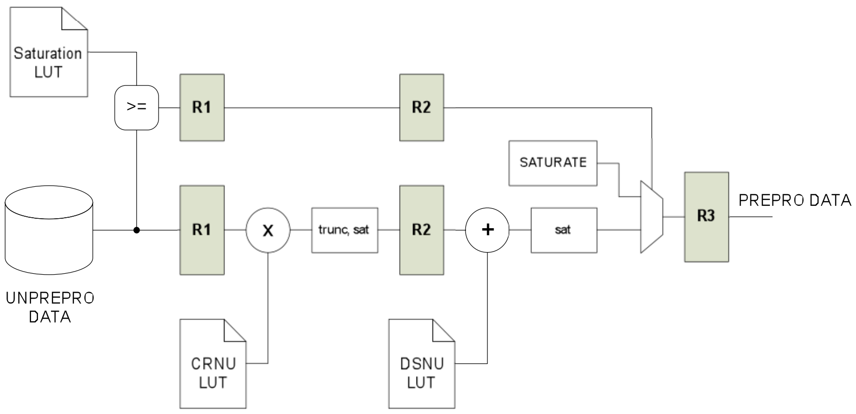
- for FPGAs only
- config interface = single point of failure
- how derive new configuration?
  - static => too memory intensive
    need config for each defect set
  - dynamic => too performance intensive
    need PPR tool on mission

How control Reconfiguration?

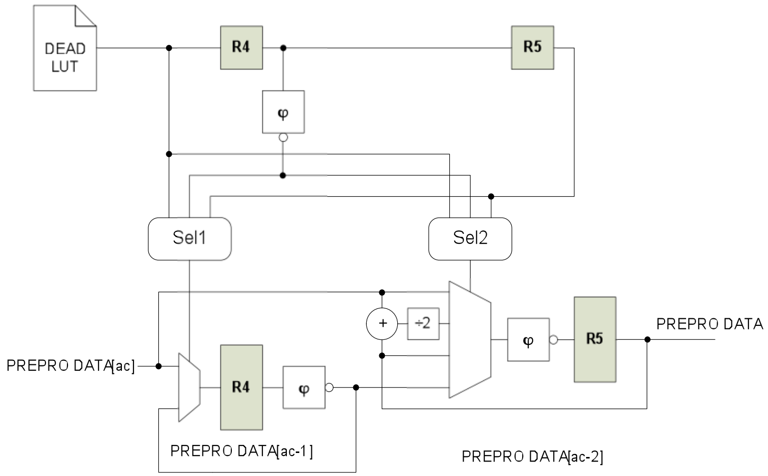
- Simple (=robust) solution: [initial idea]
  - „random repair“ without diagnosis
  - bits of a counter control switches
  - count up upon watchdog timeout
    => new configuration
  - if defect not removed => circuit still halted
    => next timeout => new try
  - with first valid configuration circuit operation continues

Application Study: GAIA VPU

Part of the video processing algorithm used in the ESA space mission GAIA
GAIA VPU = GAIA Video Processing Unit



linear correction



dead column correction

Why use this Application?

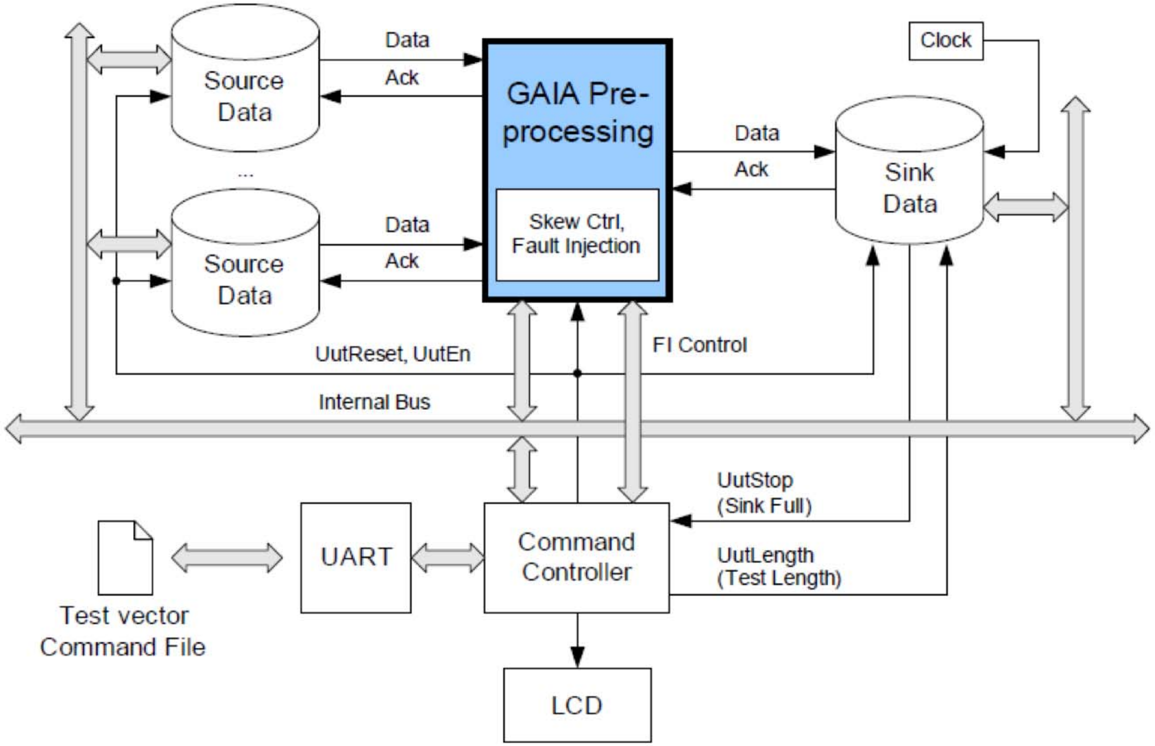
- real-world circuit structure and size
  - pipeline with forks, joins and loops
- typical space application
  - long mission time
  - extreme environment
  - high dependabiltiy required
  - no manual repair possible

    => self-healing is attractive

Environment for HW-Experiments

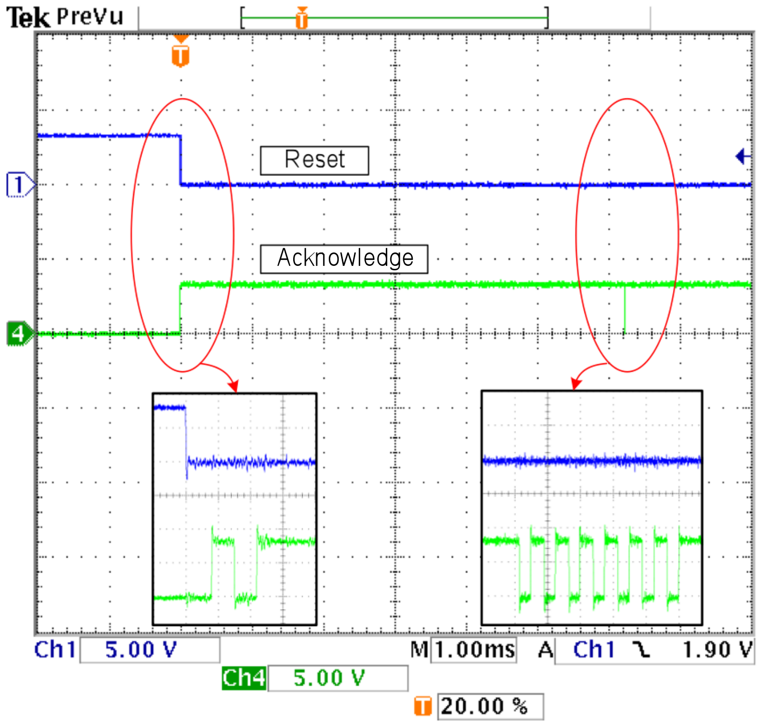
…embedded into the fault injection environment

STEFAN = Synthesizeable Test Environment For Asynchronous Networks



HW Experiments – Results

- Autonomous reconfiguration
- Single stuck-at fault injected at internal acknowledge signal
- Counter used as reconfiguration controller



HW Experiments – Resources

- # of 4-input LUTs (Xilinx Virtex-4)

	resources	relation
Synchronous GAIA	35	5%
FSL GAIA (reference)	755	100%
SH-GAIA	1565	207%
Reconfiguration Unit (RU)	39	6%
SH-GAIA incl. RU	1604	213%

- Standard FPGAs can be used for prototyping of asynchronous logic, but are not efficient
- 207% resources **but** multiple fault tolerance
- Reconfiguration Unit might have significant impact

Lessons Learnt

- In principle the idea works, BUT
- reconfiguration controller problematic
  - counter causes overhead => use LFSR
  - too many values to try => split controllers
  - ineffective repair attempts may corrupt state => need diagnosis and systematic repair
- better solution:
  - block-wise diagnosis
  - with local „random“ repair

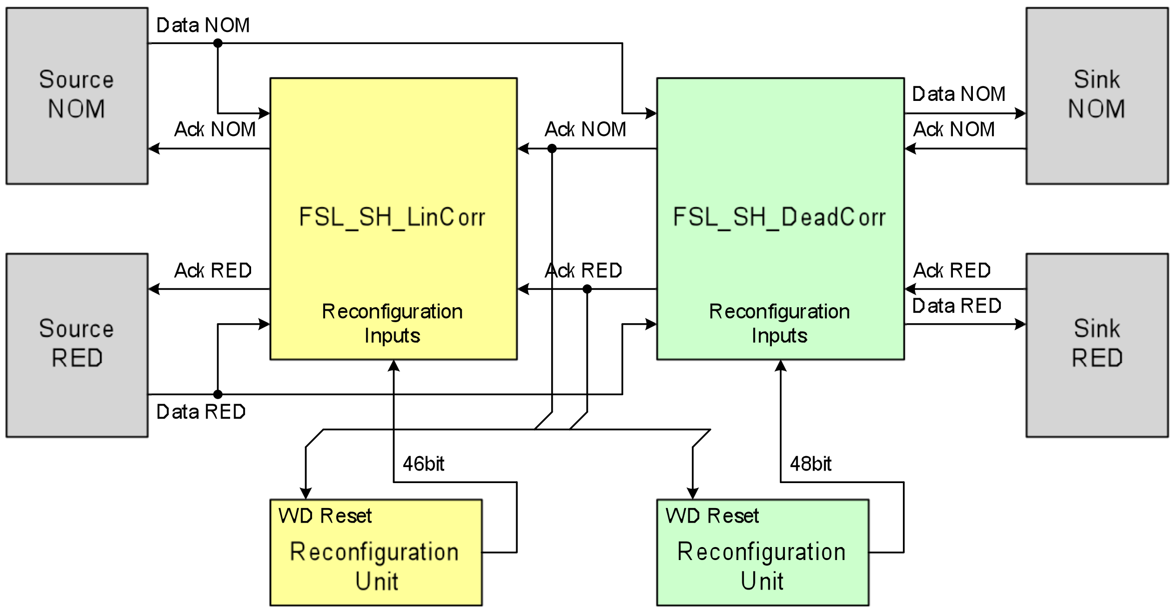
Conclusion

- asynchronous logic can solve some of the problems associated with nanoscale
- permanent faults require self-repair, asynchronous design aids in
  - detection
  - reconfiguration and
  - recovery
- fine-grain repair beneficial over component-level repair
- presented solution shown to work in principle but reconfiguration controller

Thank you for your attention!

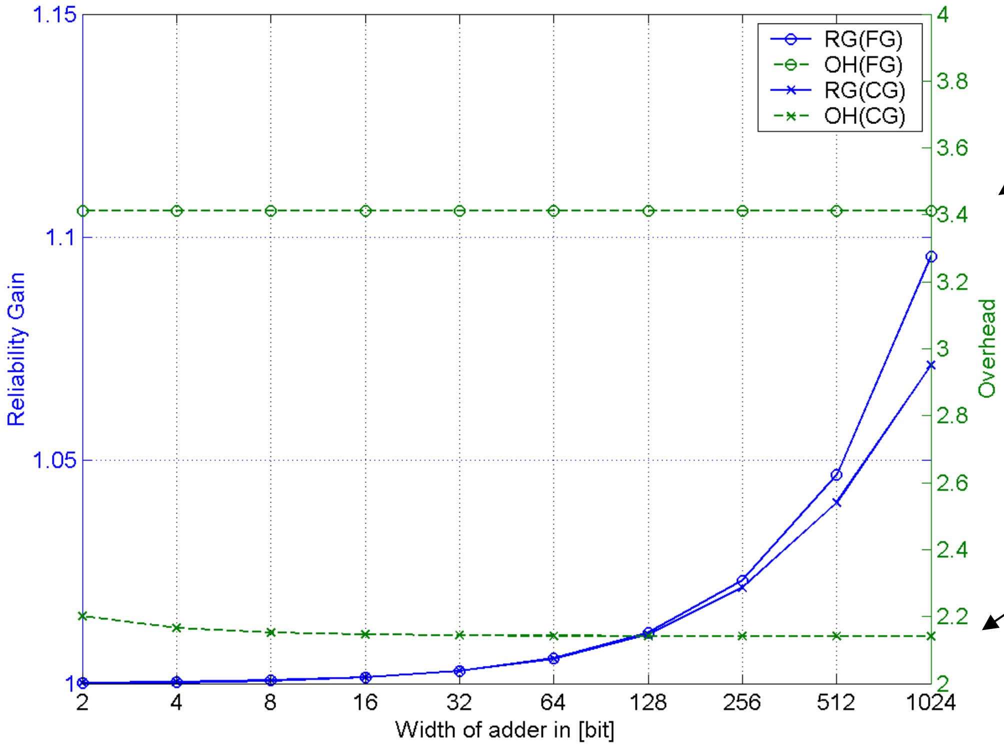
Environment for Experiments

Self-Healing implementation…



SHC Reliability vs. Overhead

Example: fine/coarse granular SHC adder



coarse grain:
constant overhead

fine grain:
decreasing relative overhead of switches