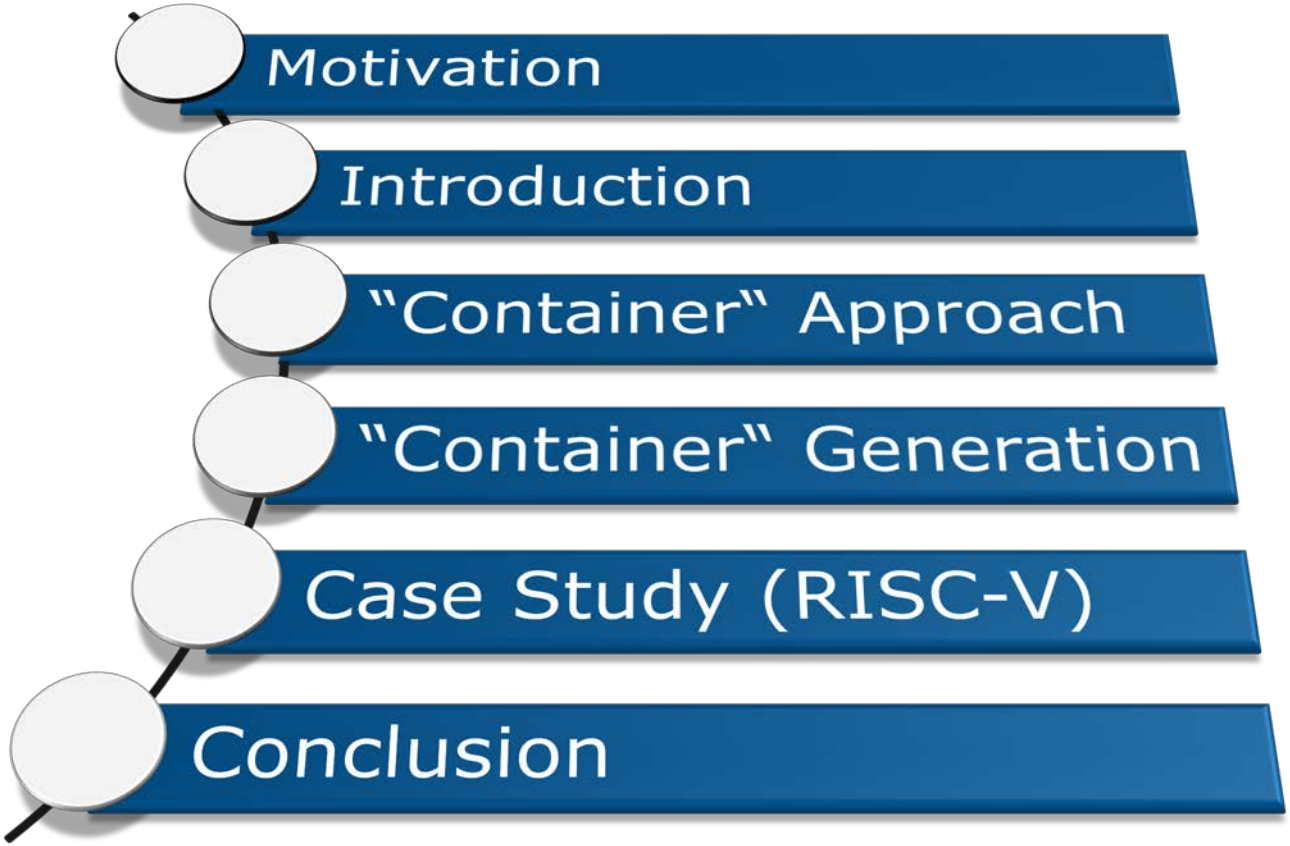
Trust is good, Control is better: Hardware-based Instruction-Replacement for Reliable Processor-IPs

Kenneth Schmitz
Arun Chandrasekharan
Jonas Gomes Filho
Daniel Große
Rolf Drechsler

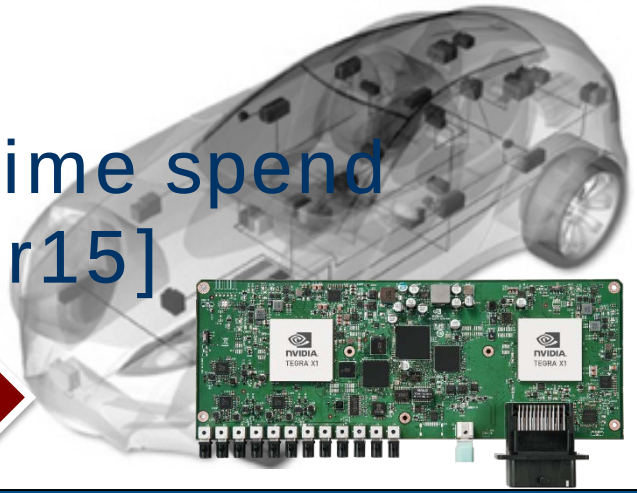
University of Bremen and DFKI Bremen, Germany
kenneth@informatik.uni-bremen.de

Outline


Motivation
Introduction
"Container" Approach
"Container" Generation
Case Study (RISC-V)
Conclusion

Motivation

- Developing highly integrated Systems
  - ➢ complex **and** challenging task
- Being ahead of competitors
  - ➢ short time to market constraints
- Also: Larger product differentiation and tighter cost margins
- Almost 60% of development time spend on test and verification [Foster15]

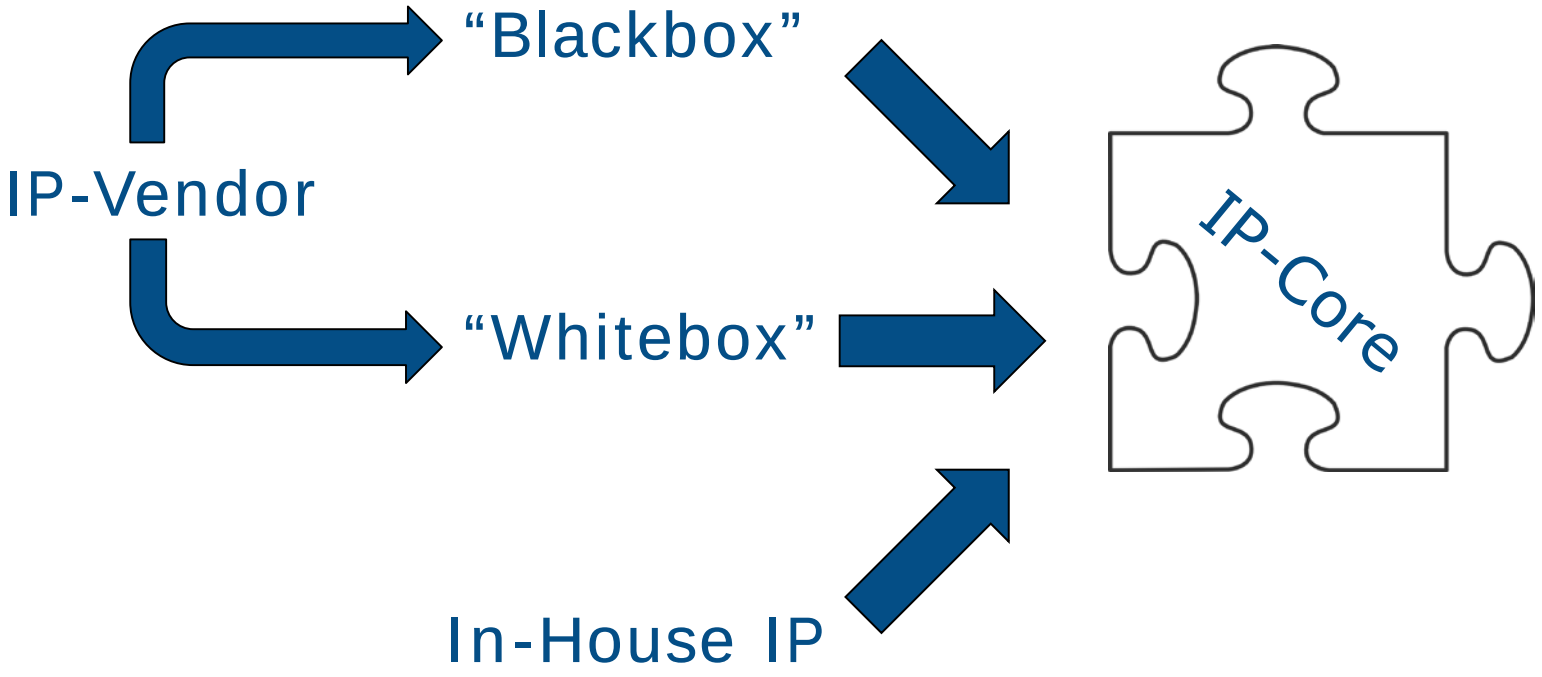


Development | Test & Verification

IP-based Design

- Often, complex systems are assembled from IP-cores
- Re-development time consuming
- Re-verification impractical due to missing sources (if third party)
- This notion of “trust” is **not** good enough for reliable systems

IP-Integration Flow



Handling Third-Party IP-Components

- System composed of building blocks which implement unified interfaces (e.g. busses)
- During integration: One not-verified instance *could* jeopardize the stability of the entire system

- Container-Solution
  - "The outside / inside is safe to handle"

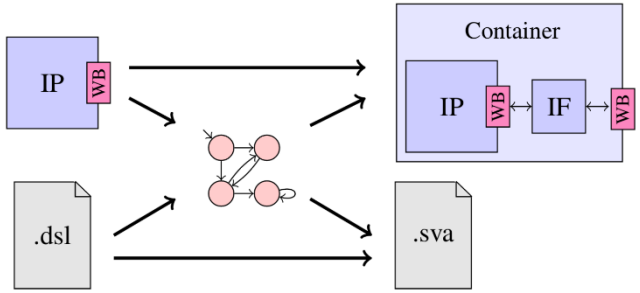
Container Principle for SoCs

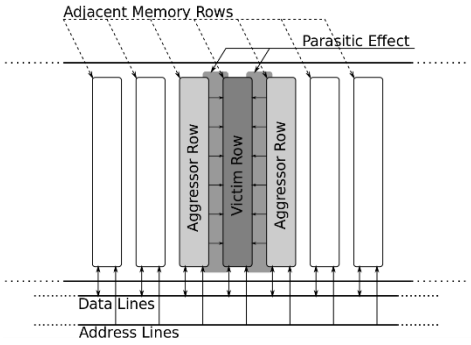
- The surrounding system is protected from the embedded component

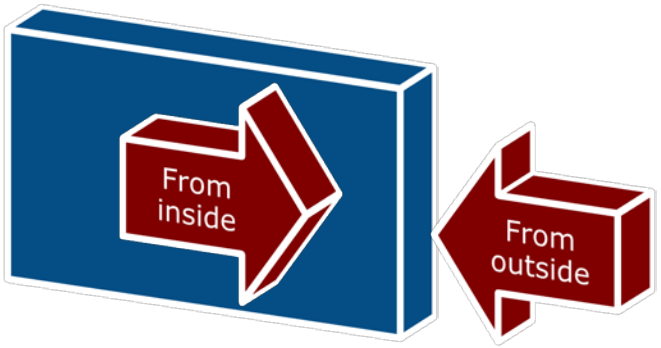
  (e.g. component stalls system bus)
  [KD2014]

- The embedded component is protected from the surrounding system

  (e.g. crypto cores leaking secret, reverse engineering)
  [CSKD15]

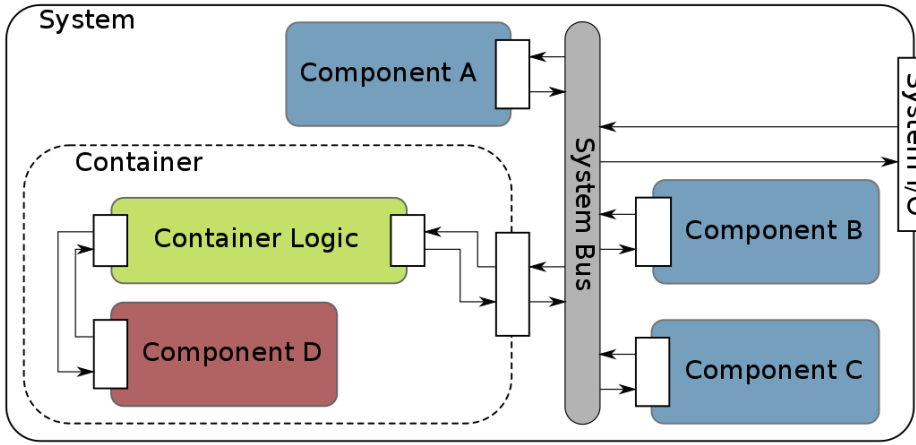






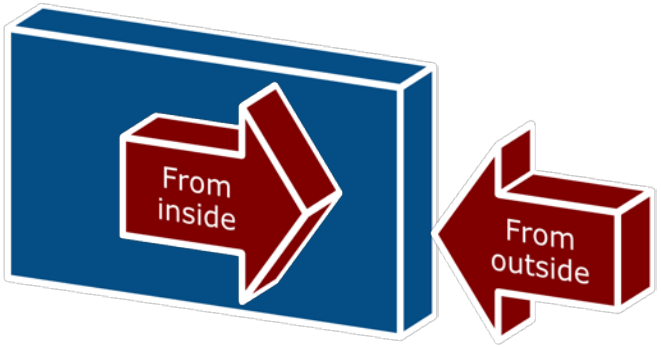
Container Principle for SoCs

Now: Instruction Replacement for blackbox RISC-Processor IP-Cores



- Transparent and Lightweight approach
  (for Soft- and Hardware)

- Flexible and Extensible



Container Generation

- **Objective**
  - *Requirement*

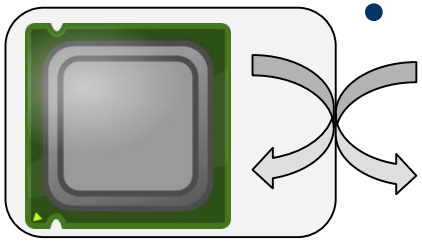
    "Systems must be verified"
  - *Consequence*

    "Verifiable portion must be small"
- The container introduces little additional logic and is constructed from a DSL correct by construction.

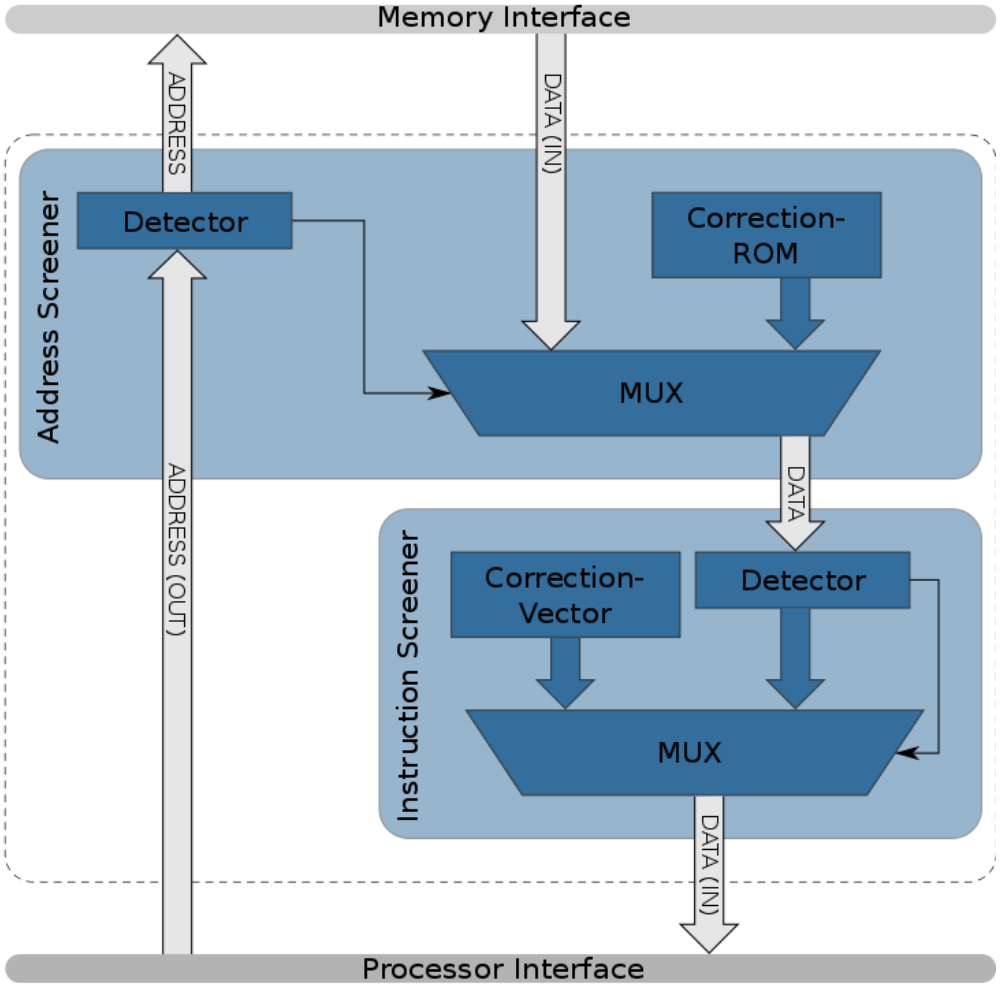
  (Strong argument, but container can be verified since it introduces little logic overhead)

(Processor-) Container Generation

- **Idea**
  - Instruction replacement prevents erroneous executions
  - Alternative approaches correct response of the system or require more insight
  - When established:
    - Approach completely transparent from software perspective (and blackbox-capable)



(Processor-) Container Architecture



- Instr. Screener
  - Detects faulty instruction
  - Feeds “jalr” to defer execution
- Addr. Screener
  - Detects deferred execution
  - Feeds substitution
  - Returns to regular execution

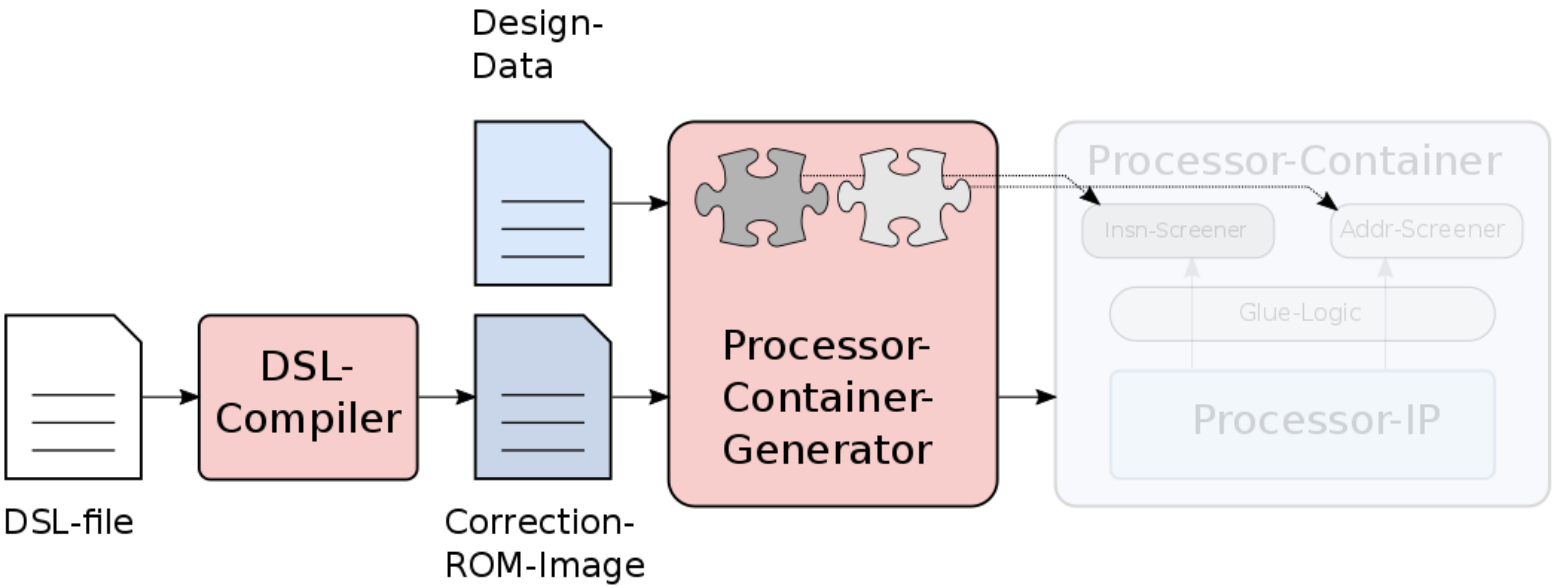
(Processor-) Container Description

- DSL Template

```
DETECT: /*erroneous instruction*/
CORRECT_BEGIN:
      /*replacement code segment*/
      /*Procedure :
            check assembly code
            backup assembly code
            alternate compute assemblycode*/
CORRECT_END:
```

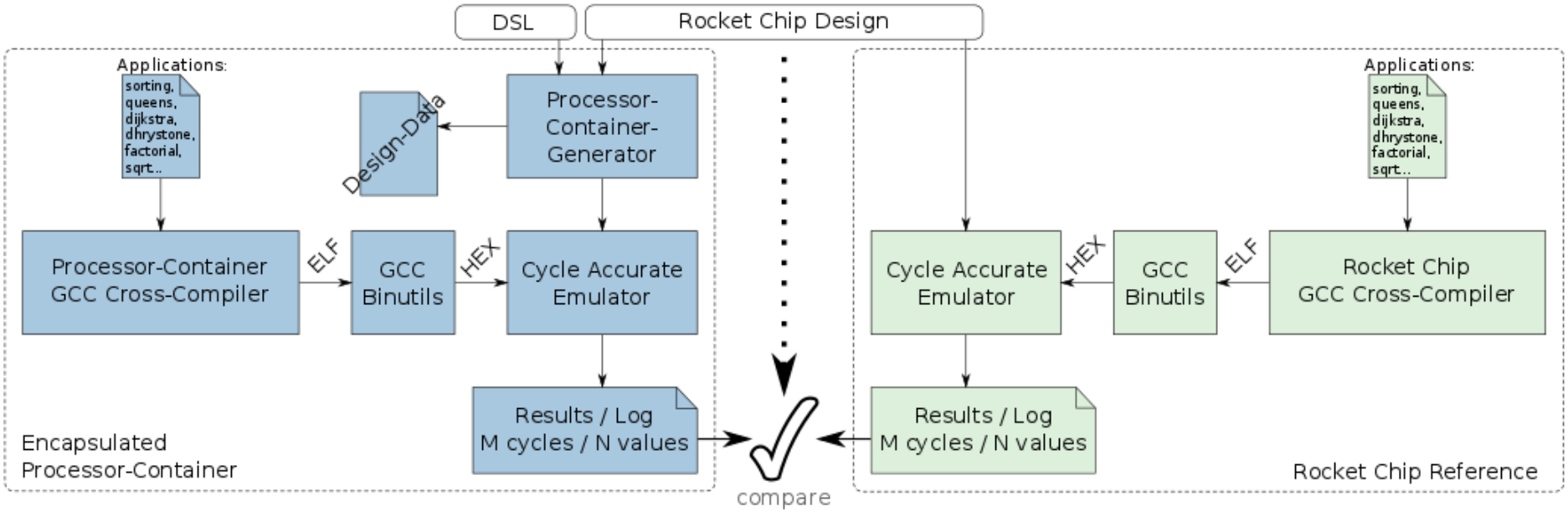
Container Generation Flow

- Input: DSL-File, Design-Data



- Output: “Hardened” (more robust) System

Experimental Setup



- RISC-V (with / without container)
- Ported libraries (newlib)
- Same benchmarks in both executions
- Chisel-generated cycle accurate emulator

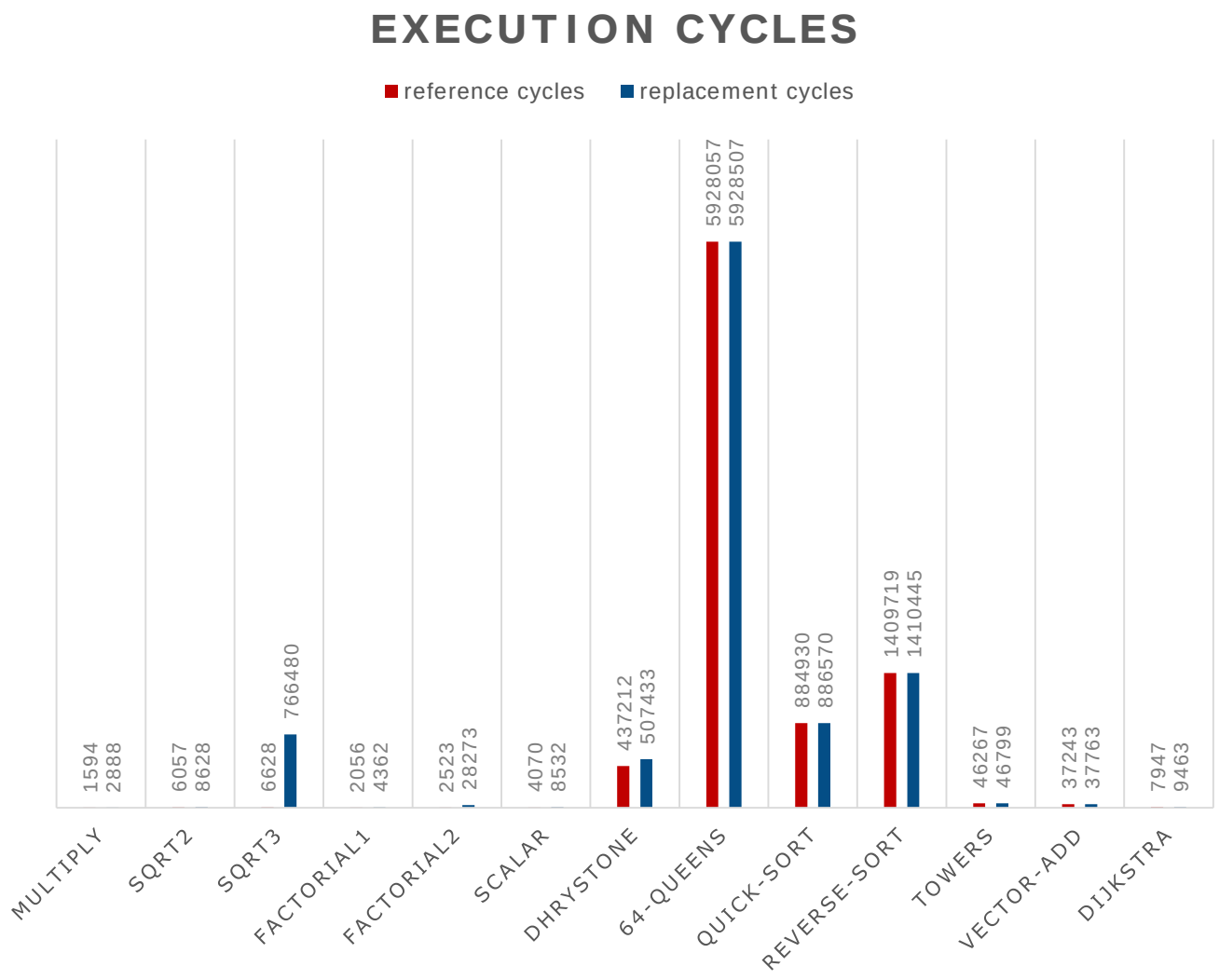
Results

- Algorithms / Programs

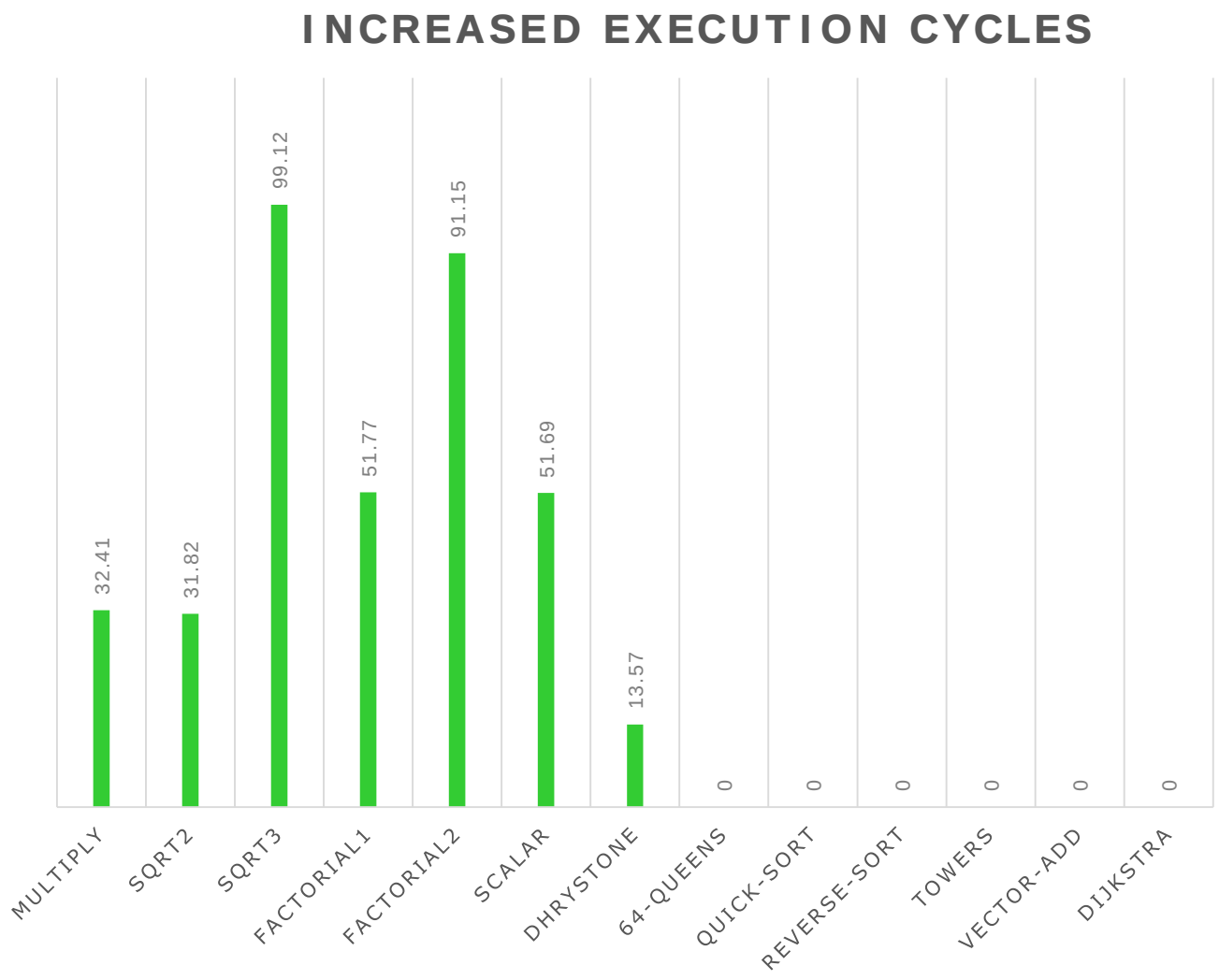
  (square-root, Dhrystone, sort, etc.)

  1. Number of cycles (reference / replacement)
  2. Increase factor

Results I


EXECUTION CYCLES
reference cycles
replacement cycles
1594
2888
6057
8628
6628
766480
2056
4362
2523
28273
4070
8532
437212
507433
5928057
5928507
884930
886570
1409719
1410445
46267
46799
37243
37763
7947
9463
MULTIPLY
SQRT2
SQRT3
FACTORIAL1
FACTORIAL2
SCALAR
DHRYSTONE
64-QUEENS
QUICK-SORT
REVERSE-SORT
TOWERS
VECTOR-ADD
DIJKSTRA

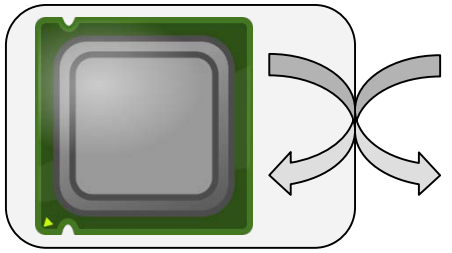
Results II


INCREASED EXECUTION CYCLES
32.41
31.82
99.12
51.77
91.15
51.69
13.57
0
0
0
0
0
0
MULTIPLY
SQRT2
SQRT3
FACTORIAL1
FACTORIAL2
SCALAR
DHRYSTONE
64-QUEENS
QUICK-SORT
REVERSE-SORT
TOWERS
VECTOR-ADD
DIJKSTRA

Conclusion

- In-Place Instruction Replacement for RISC Processor-IP
- Easy-to-use IP-integration methodology
  - Simple DSL-driven application
  - Predictable speed degradation
  - Negligible logic overhead (on FPGA)

    (Registers: 0.07%, LUTs: 0.72%, Logic: 0.85%)
- Fast Verification of Container-Logic Overhead



Trust is good, Control is better: Hardware-based Instruction-Replacement for Reliable Processor-IPs

Kenneth Schmitz
Arun Chandrasekharan
Jonas Gomes Filho
Daniel Große
Rolf Drechsler

University of Bremen and DFKI Bremen, Germany
kenneth@informatik.uni-bremen.de

Benchmarks

Application	number of mulw instr.	reference cycles	replacement cycles	increase factor	correction cycles	%
multiply	20	1594	2888	1.812	936	32.41
sqrt2	24	6057	8628	1.424	2745	31.82
sqrt3	567	6628	766480	115.642	759712	99.12
factorial1	60	2056	4362	2.122	2258	51.77
factorial2	147	2523	28273	11.206	25772	91.15
scalar	120	4070	8532	2.096	4410	51.69
Dhrystone	**2005**	**437212**	**507433**	**1.161**	**68880**	**13.57**
64-queens	*0*	*5928057*	*5928507*	*1.000*	*0*	*0*
quick-sort	*0*	*884930*	*886570*	*1.002*	*0*	*0*
reverse-sort	*0*	*1409719*	*1410445*	*1.001*	*0*	*0*
towers	*0*	*46267*	*46799*	*1.011*	*0*	*0*
vector-add	*0*	*37243*	*37763*	*1.014*	*0*	*0*
Dijkstra	*0*	*7947*	*9463*	*1.191*	*0*	*0*