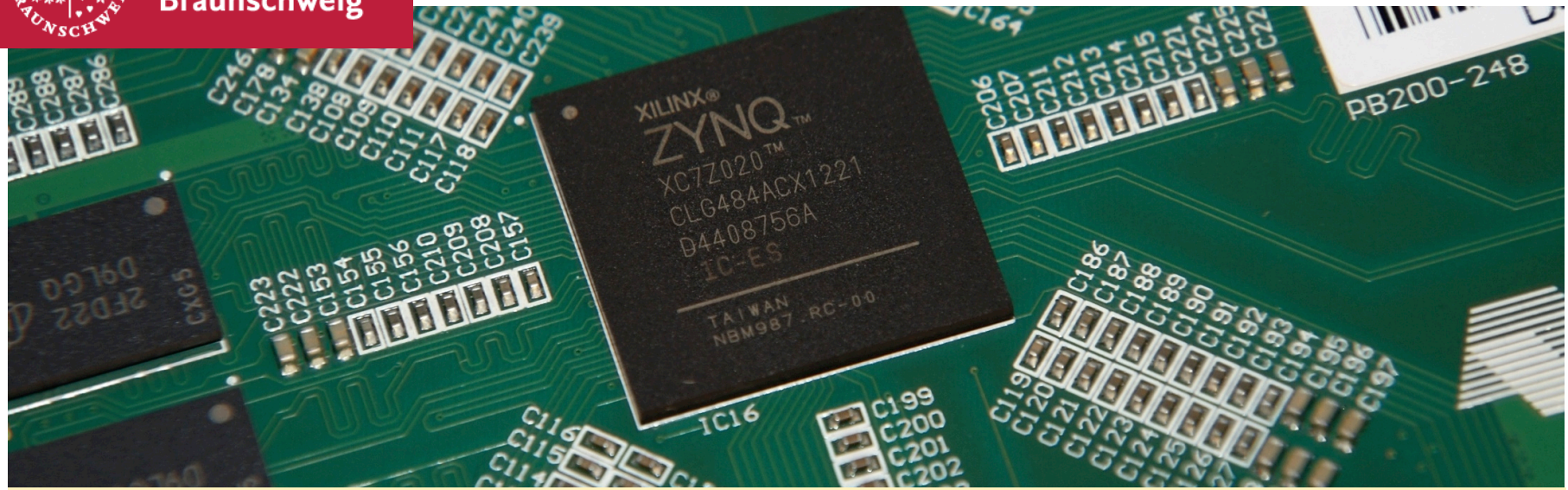
Technische Universität Braunschweig

Chair for Chip Design for Embedded Computing




# A Scriptable, Standards-Compliant Reporting and Logging Extension for SystemC

Jan Wagner, 19th July 2015

# Background

- In the past years our group created SoCRocket

- SoCRocket is
  - ESA's official virtual platform for the Leon3/Grlib ecosystem
  - SystemC 2.3 and TLM 2.0 based
  - Leon CPU (TrapGen), AHB, APB, Interrupt controller, MMU & caches, Memory controller, etc.
  - Open source: https://github.com/socrocket

➔ We identified the need for a powerful reporting system!

# Which solution does SystemC provide today?

- `sc_report`
  - Usage: `SC_REPORT_INFO("id", "message")`
  - Default handler: print to screen, send to file
  - Simple filters
  - Prepared for customization

| Name | Type |
|---|---|
| id | string |
| message | string |
| severity | SC_INFO, SC_WARNING, SC_ERROR, SC_FATAL, SC_MAX_SEVERITY |
| verbosity | int |

# What are the limitations?

1. Often additional information about the internal state is required, e.g.
   - which address is accessed
   - value of a certain register
   - type of access

➔ Traditional solution: convert information to string and append it to the message

Example:

```
std::stringstream message;
message << "APB slave configuration addr=" <<
  std::hex << apb.get_base_addr() <<
  " size=" << apb.get_size();

SC_REPORT_INFO("/example/id", message.str());
```

# What are the limitations?

2. The `SC_INFO` level with high verbosity produces too many messages
   - Hard to find the important messages
   - Simulation will be very slow
   - Possible solutions:
     - Use filters of the standard report handler (based on string comparison, component instances not addressable)
     - Use grep (slow on large files)

# What are the limitations?

3. Code of models and reporting control are not separated
   - Hard to maintain
   - SystemC source needs to be altered
     - Behavior of conditional reports is hard-coded in the models
     - Almost impossible to provide long-term support for different analysis scenarios
     - At least new linking required

# What do we propose to make reporting better?

1. Add the possibility to append **key-value pairs** to each report

2. Filters based on **numerical object-ids** instead of string comparisons

3. **Script-based report handling**

# 1. Key-Value pairs

- Add an arbitrary number of key-value pairs to reports
- Default ID is `sc_object.name()`

Example:

```
#include "sr_report.h"

// ...

srInfo("/configuration/gptimer/apbslave")
      ("addr", apb.get_base_addr())
      ("size", apb.get_size())
      ("APB Slave Configuration");

// ...
```

# 2. Filters based on numerical object-ids

- Filters decide only on numerical inputs
  - Numerical object-id (`sc_object` address)
  - Severity
  - Verbosity

➔ Integer comparisons are much faster than string comparisons

➔ `sc_object` base addresses are explicit compared to string-ids as proposed for SC_REPORT

# 3. Report handling with scripts

- Easy to adapt to different use-cases
  - Less code than in C++
  - User-friendly API
  - Extensive libraries in target language
  - Optimization friendly implementation

Without altering the SystemC code different analysis-scripts can be applied

**➔ Ideal for rapid-prototyping**

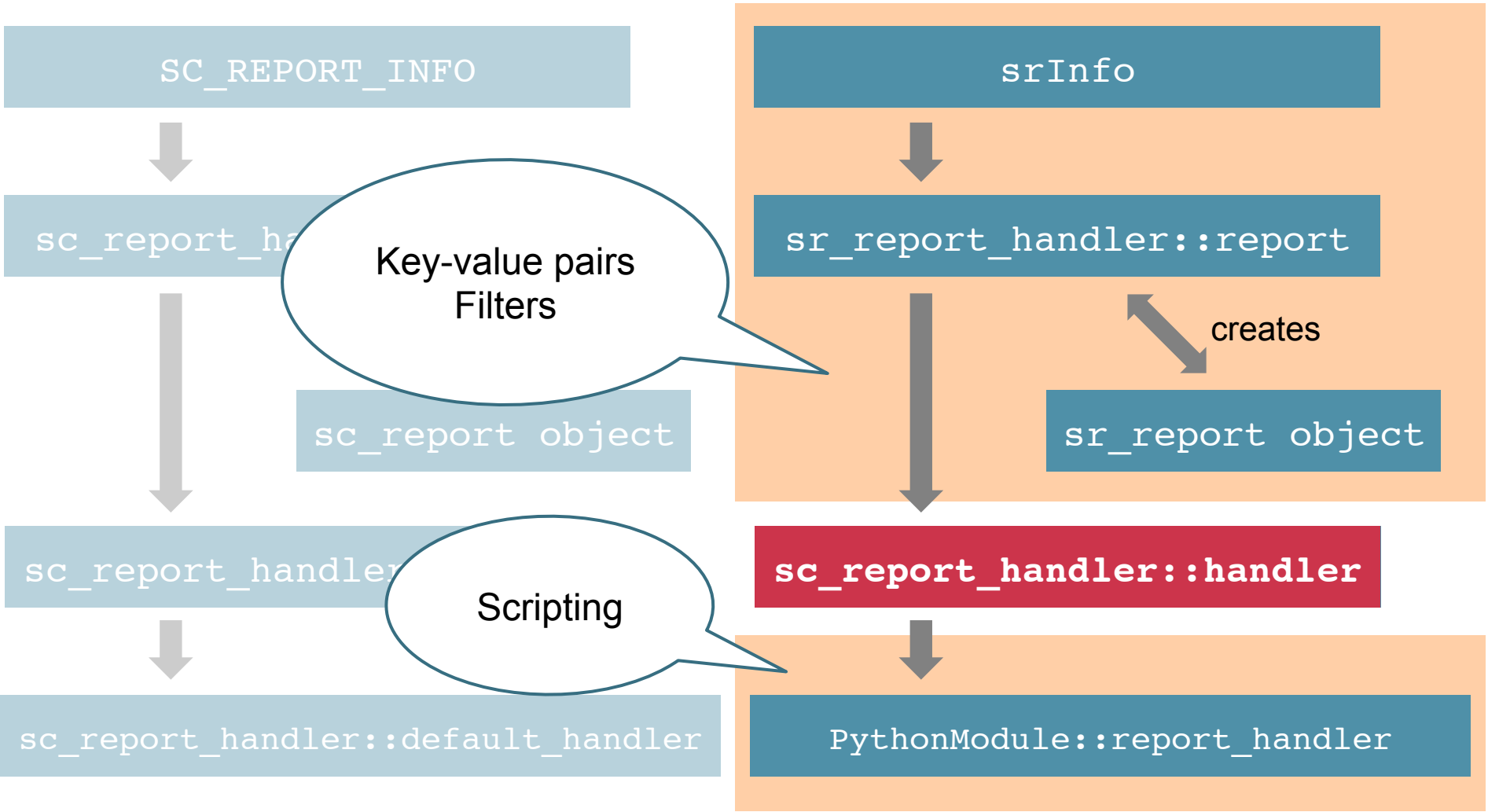
**But:** Nothing comes for free... Speed is an issue, we are addressing it with efficient filters

# 3. Report handling with scripts

- Dynamic filters
  - Time triggered
  - Event triggered
- Various output backends
  - HDF5 database
  - MongoDB database
  - Plain text file
  - Console
- Dynamic simulation control
  - Time triggered
  - Event triggered

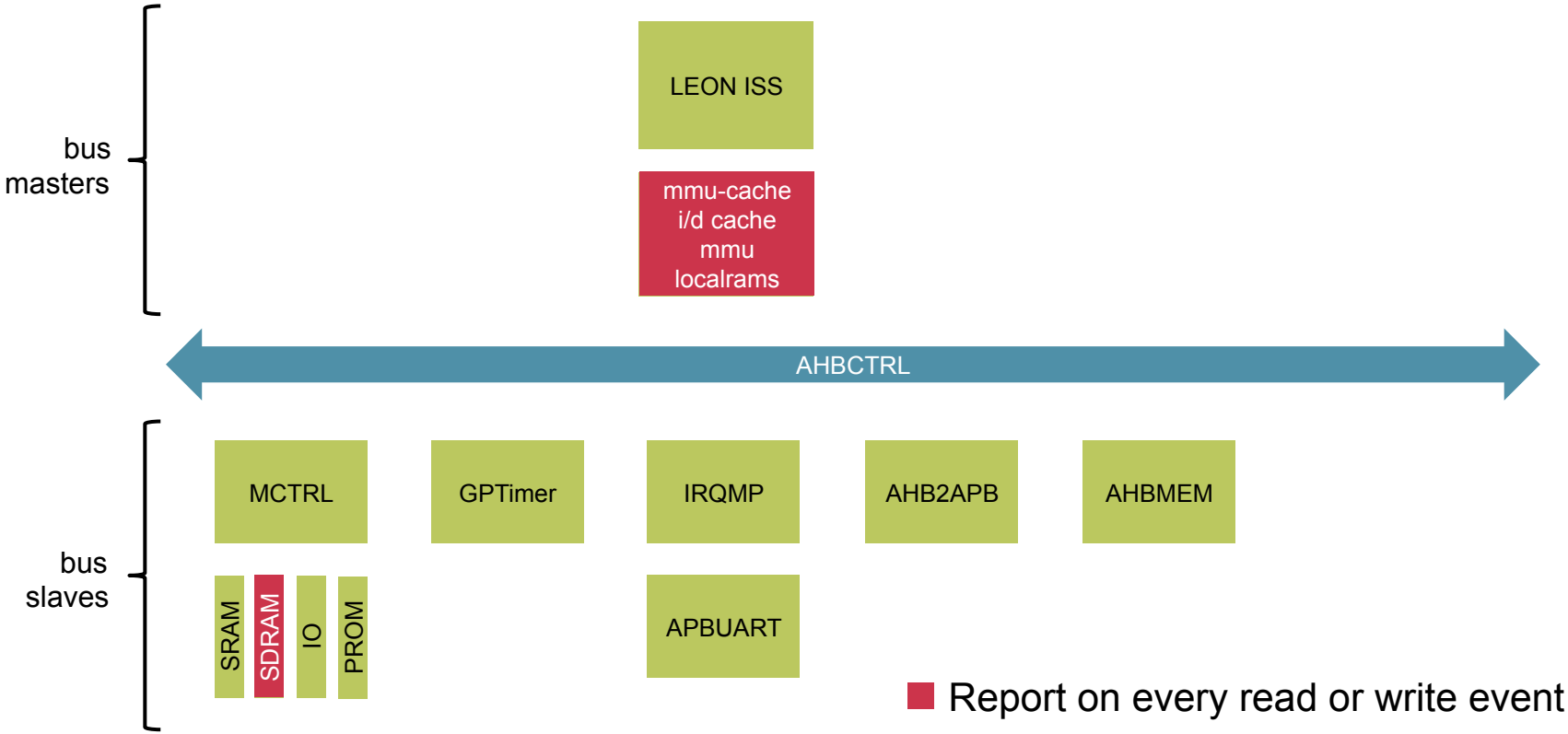
```
@sr.on(report)
def report(
        message_type=None,
        message_text=None,
        severity=None,
        file_name=None,
        line_number=None,
        time=None,
        delta_count=None,
        process_name=None,
        verbosity=None,
        what=None,
        actions=None,
        phase=None,
        **kwargs):
    pass
```

# sc_report vs. sr_report architecture



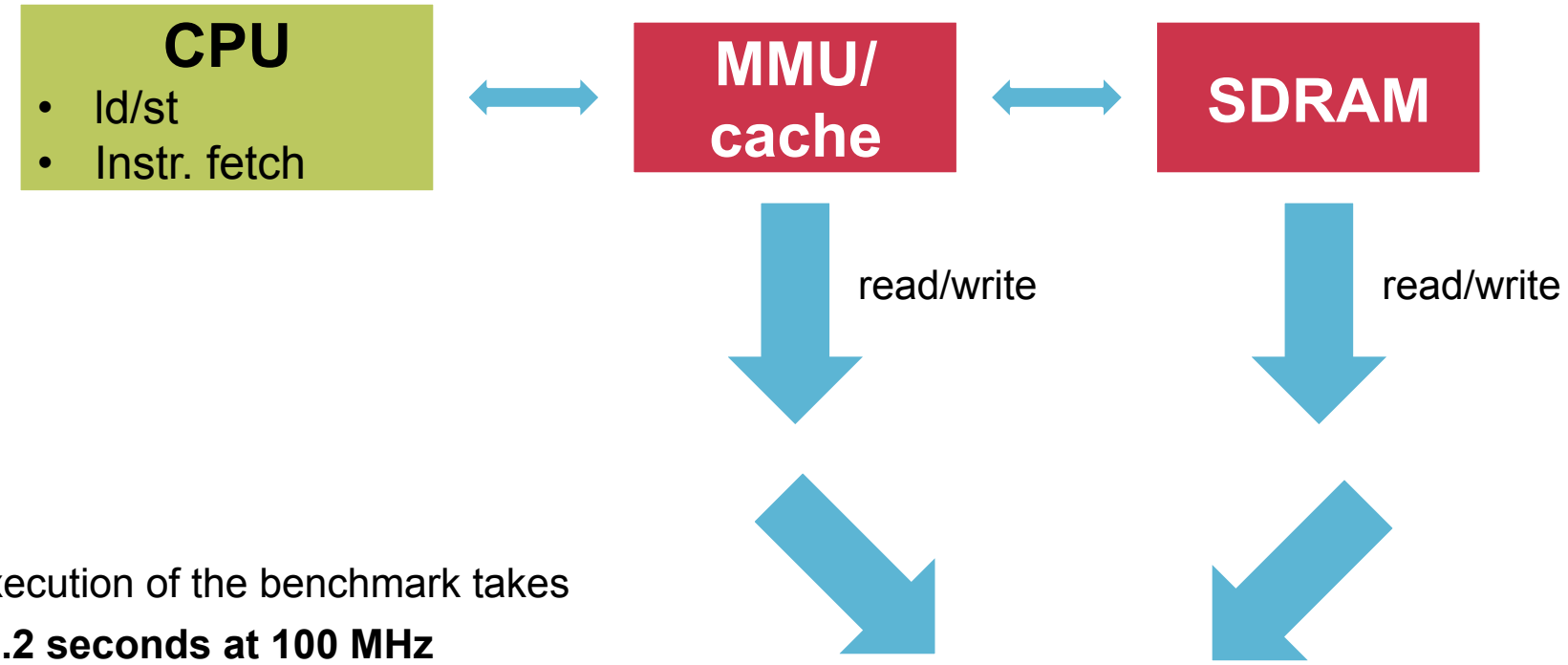
# What about performance?

The test setup:



# What about performance?

The test setup:

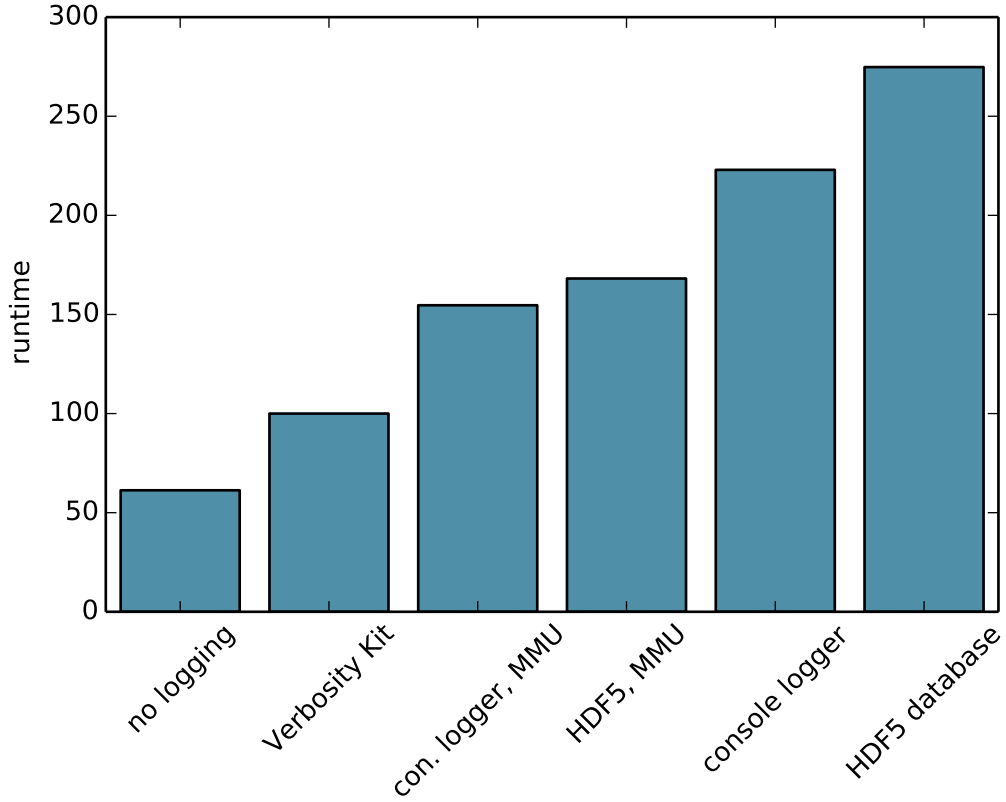



Execution of the benchmark takes
**48.2 seconds at 100 MHz**
according to the simulator.

**> 211 million reports**

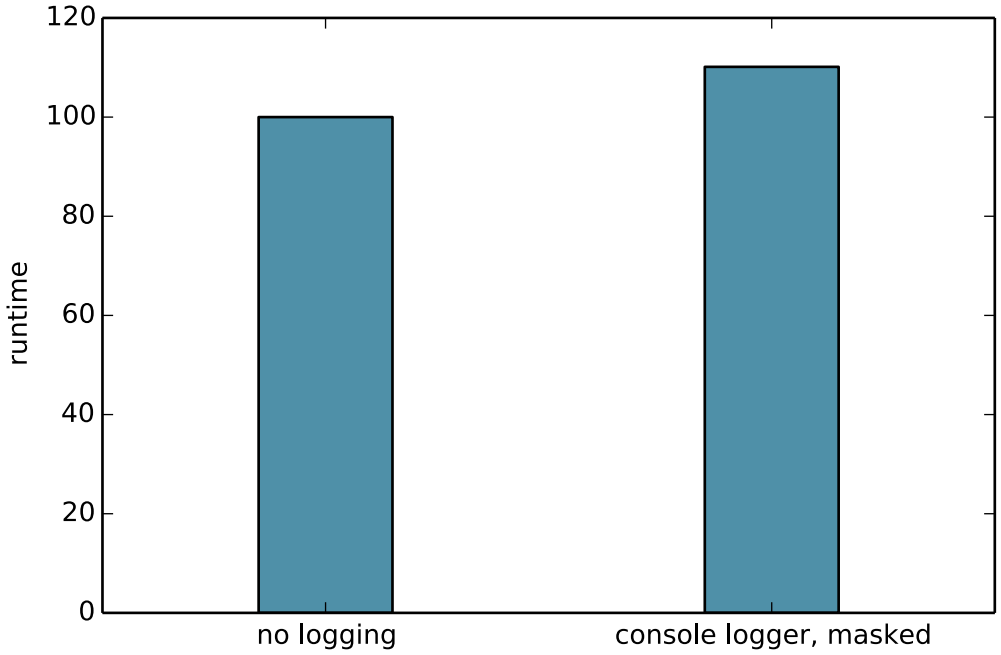
**24 GB as plain text**

# Stream-based vs. script-based reporting




- Script-based report handling takes time
- The effect can be mitigated using the proposed filters

# How efficient are the numerical filters?




- All reports supressed by filters
- Only 10% overhead

# Conclusions

- New features
  - ✓ Key-value pairs can be attached to reports
  - ✓ Efficient filters implemented
  - ✓ Flexible script-based report handling implemented

✓ Tried and tested in a real project (SoCRocket)

✓ Backward-compatible

- Future work
  - Decoupling of SystemC and script-based report handling to improve speed.

# Thank you for your attention.

# Questions?

Dipl.-Ing. Jan Wagner
wagner@c3e.cs.tu-bs.de