Tracing in TLM based SystemC models

Rocco Jonack, Eyck Jentzsch
MINRES Technologies GmbH

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by **MINRES GmbH** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

Company

- MINRES is an enabling company, dedicated to providing the expertise and solutions required for improving development productivity:
  - Methodology services + productivity IP
  - Design consulting services & value added application engineering
  - ISO26262 compliant RISC-V RTL IP
  - Cloud based hybrid simulation solution (RTL/VP)
- MINRES Technologies GmbH is a privately-held, remote-first startup based in Germany

INTRODUCTION

Purpose of traces

- Traces are commonly well understood
  - Accepted mechanism for debugging, performance analysis and documentation
- Traces are very widely used in HW design and verification
  - All HDL simulators support tracing implicitly
- SystemC contains default implementation for VCD tracing

Usage of traces

- Tracing signal type elements is straight forward
  - Bool, integer, bit constraint types and structs of those base types
  - VCD is the default choice, but many compressed formats exist
  - Open-source availability is important in many cases
- Tracing more complex elements like transactions is cumbersome in VCD
  - Transactions have many elements and time points
  - Phases as basic elements of transactions can overlap
- Why transaction tracing
  - Well formed format for transactions
    - Forms definition of data record with different timing points on a stream
    - Information on a stream can vary over time
  - Ties into TLM2 modeling concepts
    - Extension mechanisms can be transformed into trace structures
  - Can augment VCD

TRACING IN SCC - USE MODEL

Module sysc: waveform tracing

- SCC contains tracer module to automatically trace signals, ports, and variables
- Improved waveform tracing implementation(s)
  - Push waveform tracing implementation
  - Supports VCD
  - Supports FST
    - Compact format coming with gtkwave
  - Can be visualized using gtkwave, Impulse, or SCViewer

Module sysc: VCD push versus poll

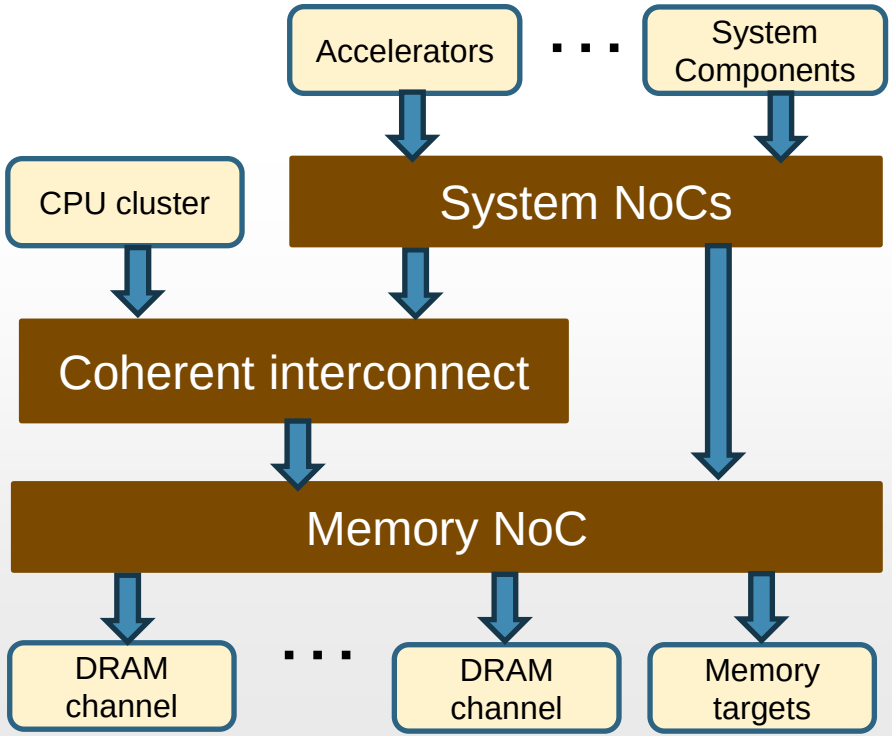
- Default implementation polls each traced signal for changes
  - This is usually called pull approach
- SCC comes with an implementation which registers a method to each signal or port to be notified when changed
  - This we call push approach

Module sysc: transaction recording

- Two implementations
  - Based on SCV
  - Based on Lightweight transaction recording (LWTR)
- Both support various backend implementations
  - Text format based on SCV reference implementation
  - Fast Transaction recording (FTR) a compact binary representation with compression
- Can be visualized using SCViewer or Impulse
- Various reference implementations read FTR format for further analysis

Transaction recorder

- Unit with 1 input and 1 output port
  - TLM2 no extensions; AXI, ACE and CHI with extensions
- Optional transaction tracing
  - Tracing is enabled if file handle was opened before construction
  - Low runtime overhead
- Can be integrated in existing systems
  - Integrator to include components
  - Top level must support file handle opening



Tracing setup

- Tracer automatically traverses object hierarchy
- Tracing is controlled by CCI parameters
- Opening database replaces default implementation
  - Registering values equivalent to default implementation

```
#include <scc/configurable_tracer.h>
#include <scc/configurer.h>

int sc_main(int argc, char* argv[]) {
    // simple configuration
    scc::configurer cfg(“system.yaml”);
    scc::configurable_tracer trace("tgc_tb_rtl", scc::tracer::FTR, true, true);
    ...
    sc_core::sc_start();
}
```

Tracing control

- Configurable tracer can control tracing by hierarchy based on CCI parameters
- Optional argument to database constructor can control time window of tracing

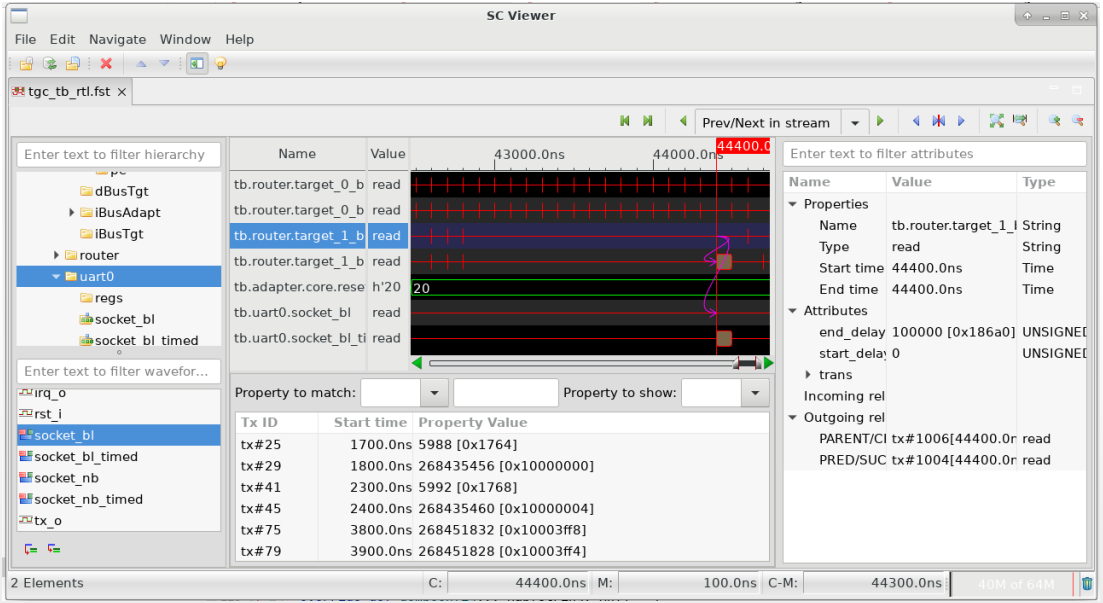
```
#include <scc.h>

int sc_main(int argc, char* argv[]) {
…
   sc_core::sc_trace_file* trc = scc::scc_create_vcd_trace_file(“my_vcd",
      []() -> bool {
        // start tracing after 2us
        return sc_core::sc_time_stamp() >= 2_us;
      }
   );
…
   sc_core::sc_start();
}
```

TRACING - VISUALIZATION

Trace visualization I

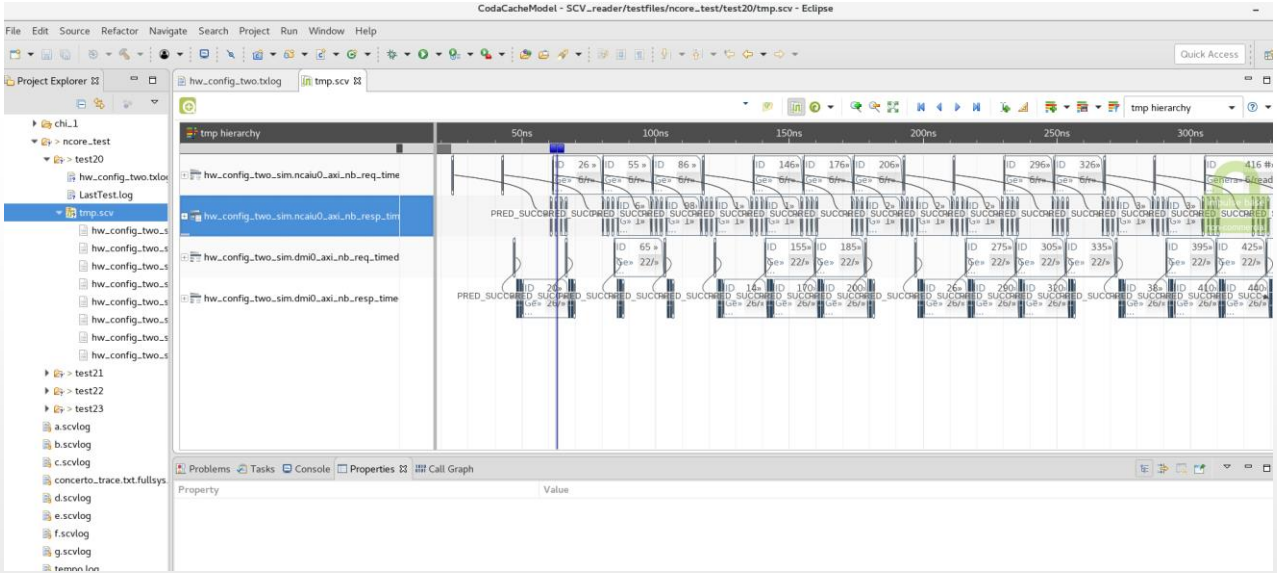
- SCViewer available on github as open-source
- Reads transaction and signal traces
- Contains waveform, table, details and hierarchy view



https://github.com/Minres/SCViewer/releases

Trace visualization II

- Impulse as open source tool for visualization
- Plugin to eclipse
- Variety of input formats
- Commercial usage and integration based on licensing

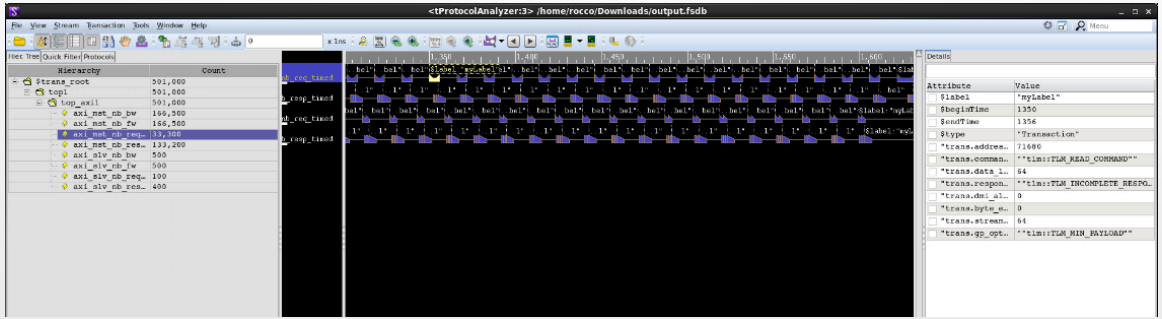


Traces visualization - gtkwave

- gtkwave is the default choice open-source tool for signal trace viewing
- Reads fast and efficiently signal traces
  - Specifically fst format
  - Provides utilities to convert from and to VCD

Traces visualization - Verdi

- Verdi is a common tool in the marketplace
- Proprietary transaction format
  - Converter from open-source transaction format to proprietary format through FSDB API
- Alignment with RTL analysis

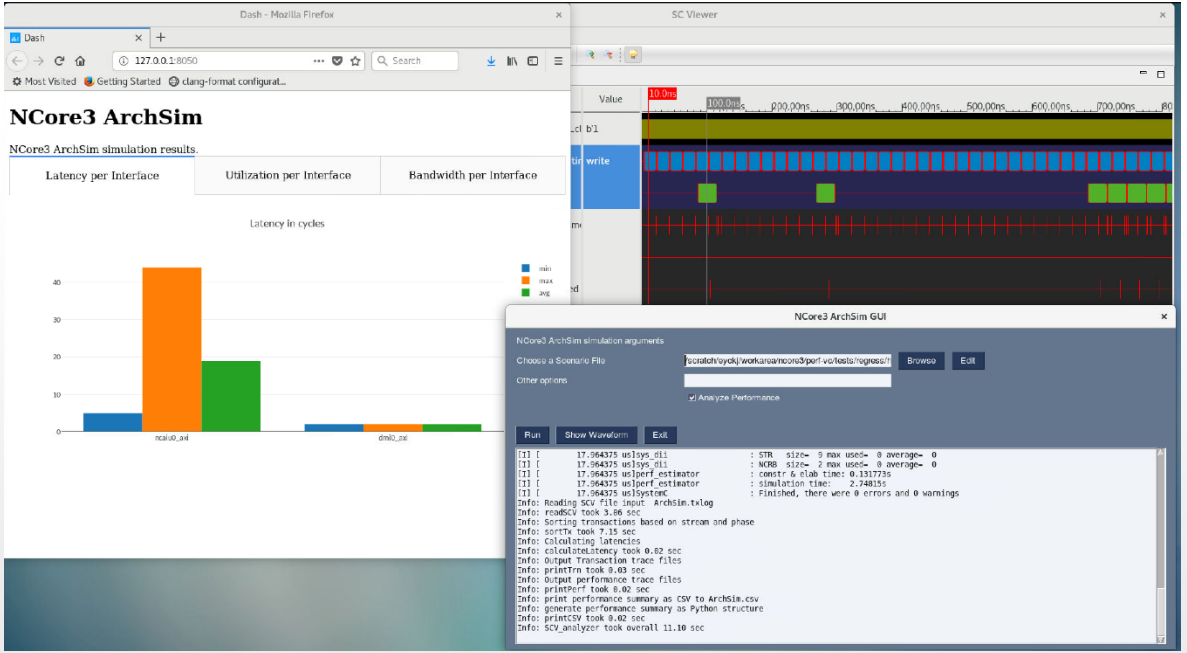


Transaction trace analysis

- Parsing and postprocessing of transaction traces
  - Well defined interfaces allow reliable post processing
  - Split transaction recording from analysis step
    - Post processing without simulation impact
    - Partial transaction analysis
    - Compression handling
    - Flexible output format
- Python script implementation
  - Example handles AXI, ACE and CHI
  - Outputs can be transaction journal, performance summary and STL per socket

Trace analysis visualization

- Trace analysis output can be used by open-source visualization tools like dash
- Python libraries allow simple analysis and even simulation control interfaces



TRACING - SUMMARY

Results of signal trace implementations

- Comparing in columns signal traces default, without signal duplication, push interface and FST implementation
- comparing in rows impact on simulation time, size of generated file and lz4 compression (where applicable)

	SystemC VCD sc_trace	SCC VCD no duplication		SCC VCD push		FST	
simulation time (s)	23,01	14,18	61,63 %	11,86	51,54 %	13,57	58,97 %
file size (Mbytes)	561,15	206,51	36,80 %	206,92	36,87 %	5,77	1,03 %
compressed file size (Mbytes)	164,33	36,20	22,03 %	38,36	23,34 %		

Results of transaction trace implementations

- Comparing in columns transaction traces as text (txlog), binary enconded (txftr) and compressed binary encoded (cxftr)
- comparing in rows impact on simulation time, size of generated file and runtime of postprocessing

	txlog	txftr		ctxftr	
simulation time (s)	163,82	121,16	73,96 %	125,00	76,31 %
file size (Mbytes)	496,89	60,06	12,09 %	8,02	1,61 %
SCV TX read time (s)	42,70	12,71	29,77 %	13,51	31,64 %
SCV TX overall (s)	49,93	18,89	37,83 %	20,35	40,76 %

Wishlist

- Alignment around transaction recording format
- Better VCD implementation as part of SystemC standard
- Alignment with formats used by RTL simulators

BACKUP