A Hybrid Channel for Co-Simulation of Behavioral SystemC IP with its Full System Prototype on FPGA

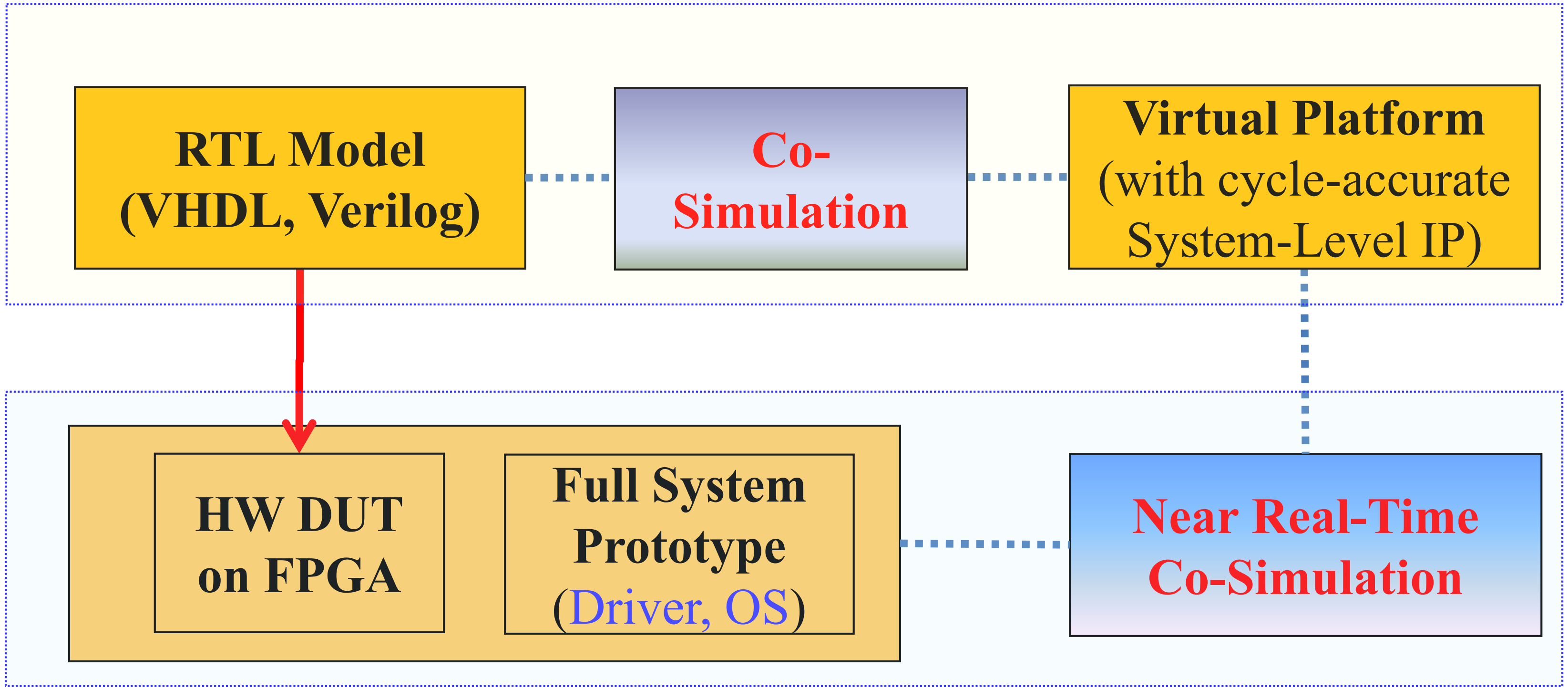
Antonis Papagrigoriou, Miltos D. Grammatikakis and Voula Piperaki

T.E.I. of Crete

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Motivation

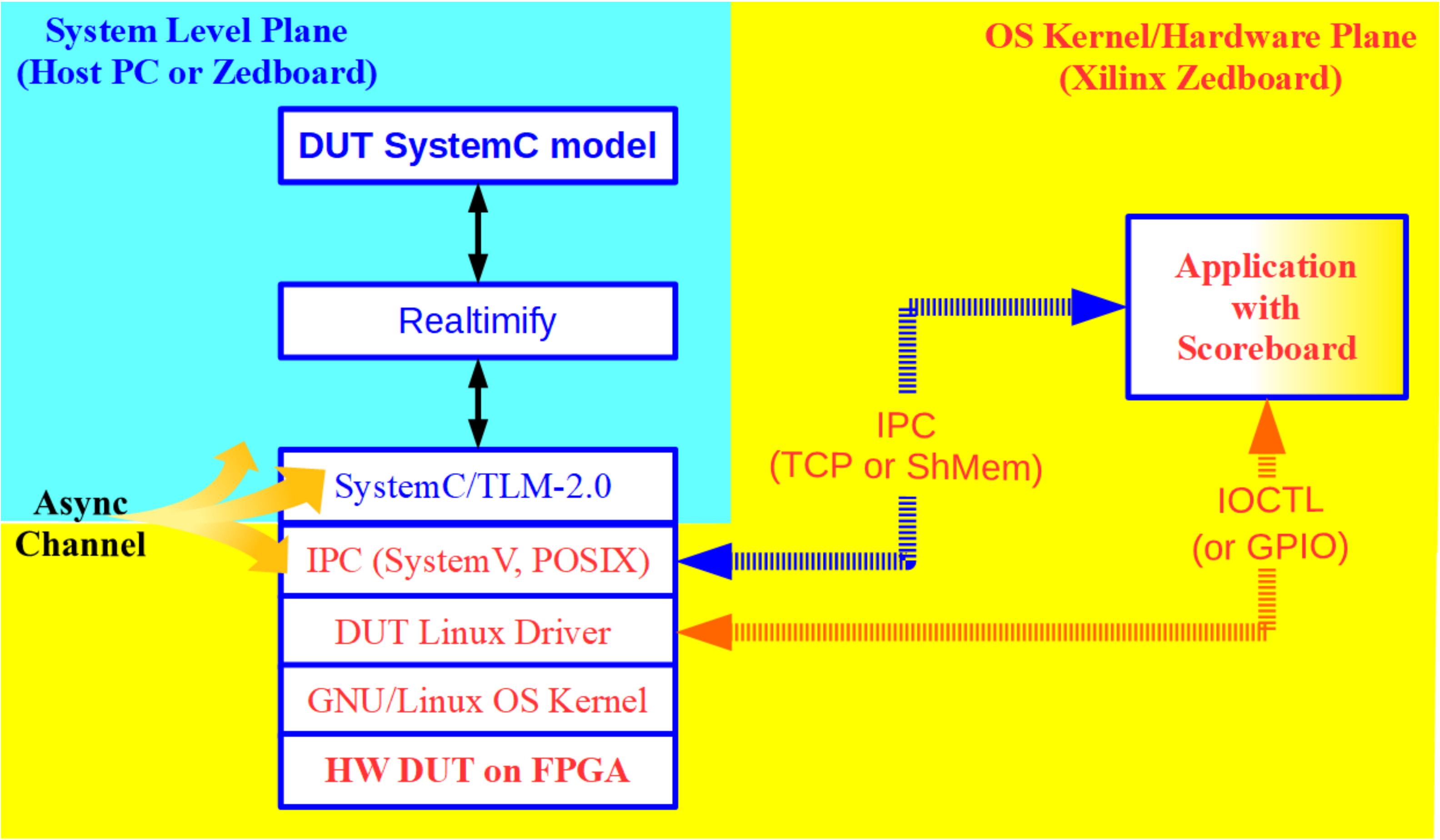


- Existing RTL – SystemC Co-Validation
  - SystemC DUT in a Virtual Platform (VP)
  - Allow VP to directly perform transactions on an FPGA-based prototype
  - Does not address full system around the FPGA prototype (only on VP side)
- Alternative near real-time command-to-command co-validation of HW DUT **prototyped on FPGA as full-system** vs SystemC DUT

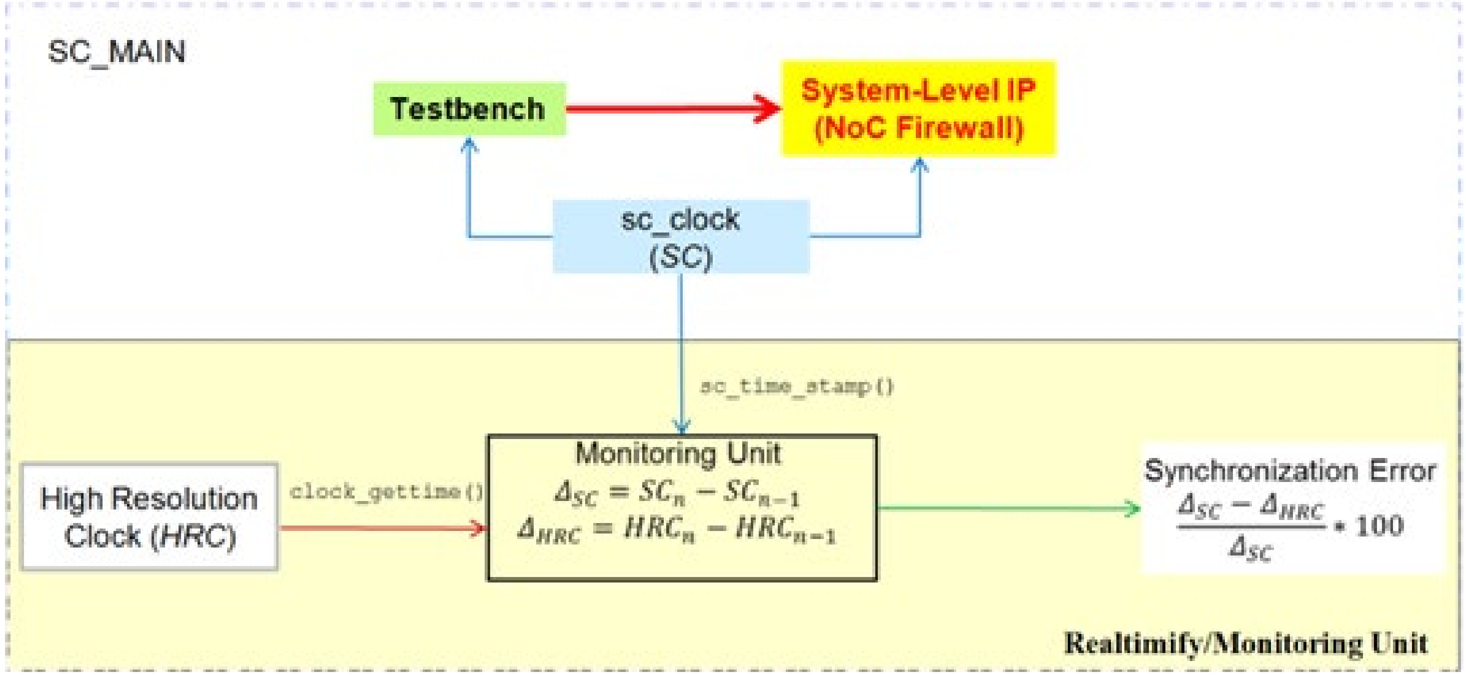
Our Co-Simulation Framework

- Provides an experimental open-source framework for co-validating a HW DUT (embedded in a real system with OS stack) against its equivalent SystemC model
- Implements a non-intrusive asynchronous SystemC channel based on Shared Memory or TCP socket primitives
- Supports a near real-time option, using the Realtimify SystemC module that allows matching simulation time with real-time execution

Proposed Co-Simulation Framework



SystemC Realtimify (Optional)



- Allows synchronizing the HW DUT and SystemC model (usleep)
- Check for correct specifications, such as performance requirements

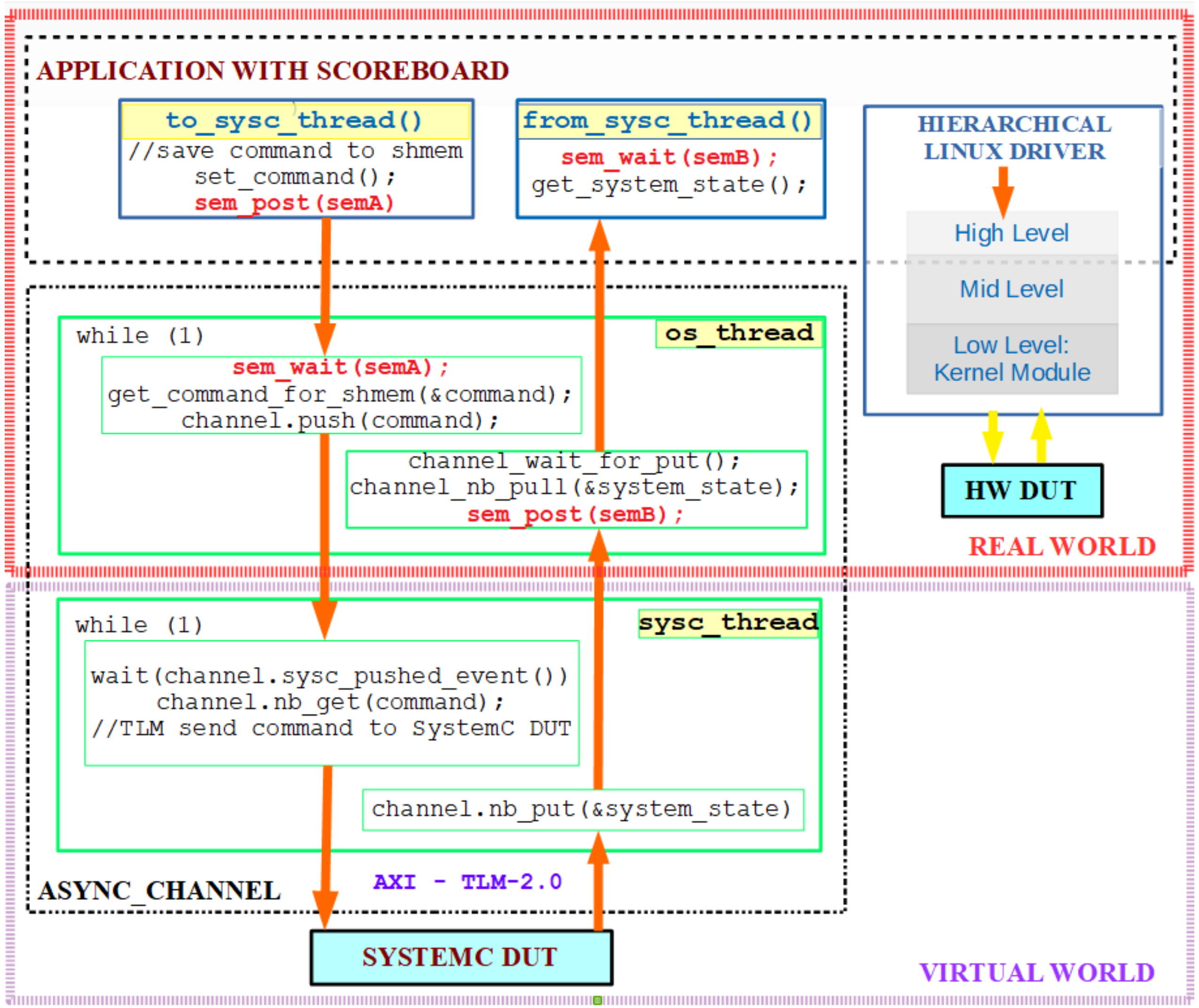
Our Asynchronous Channel

Based on an original idea by D.C. Black (DVCon 2013)

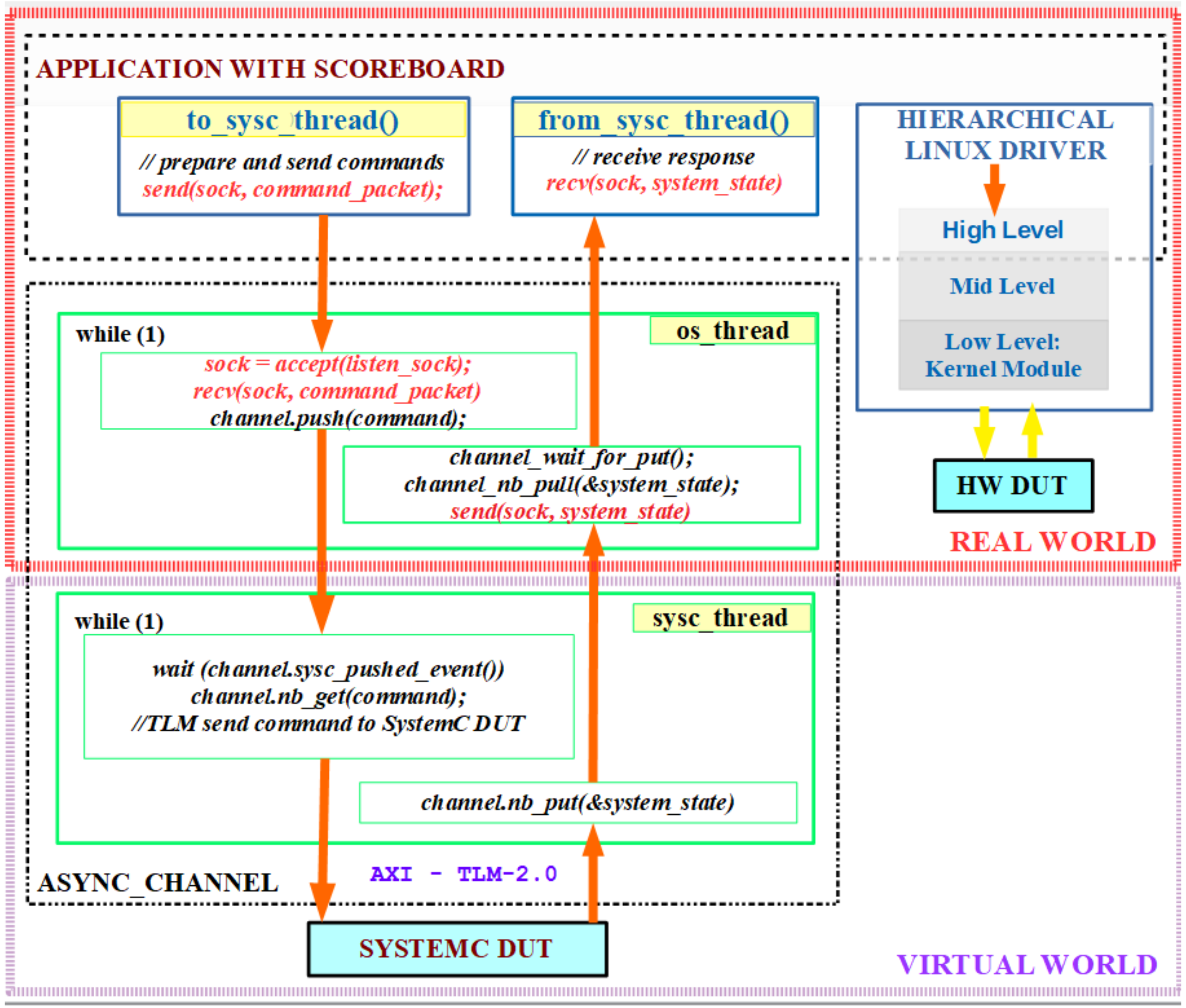
1. Propagates each command received from the application scoreboard to the SystemC DUT model
2. Returns timing, status and performance characteristics from the system-level IP back to the application scoreboard

Implemented within co-simulation platform using TCP & Shared Memory

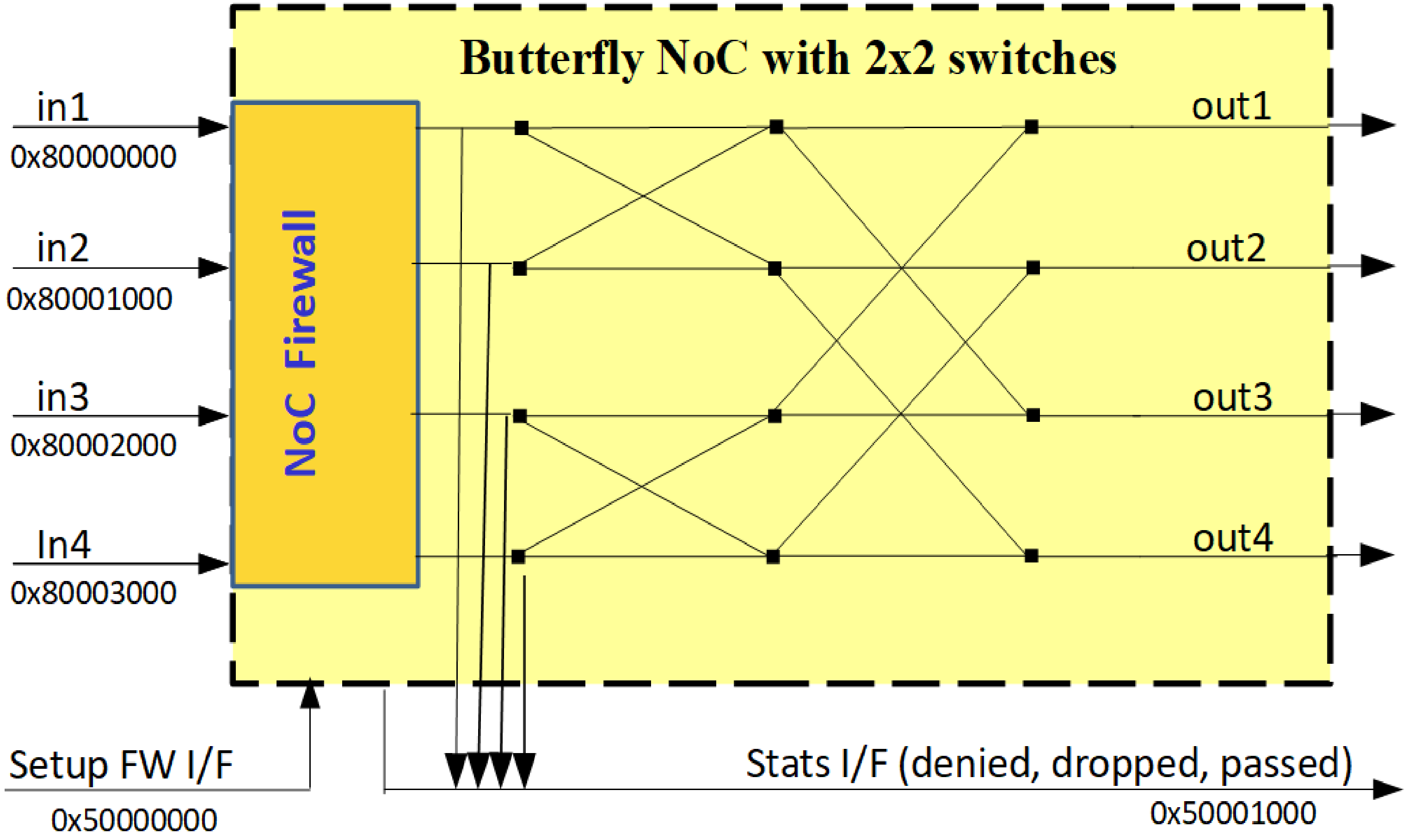
Shared Memory Implementation



TCP Implementation



DUT: NoC Firewall



DUT Drivers

Hierarchical Linux Driver for HW DUT

- High Level (HLD)
- Mid Level (MLD)
- Low Level (LLD)

Driver for SystemC model (accessed via async channel)

- Re-Implement the Low Level Driver (LLD)
- Reuse the MLD and HLD of the Hierarchical Linux Driver

Test Applications

Co-validate HW DUT and SystemC model for 3 testbenches:

1. separate read/write access, for all possible port and rule register combinations
2. industrial mHealth application focusing on healthcare data privacy
   - data protection via the NoC Firewall

All testbenches were run on ZedBoard

Testbench Complexity (HW/SW)

DUT Complexity

- SystemC model
  - ~1700 lines of code, 14 SystemC clocked threads
- NoC Firewall synthesis (Zedboard FPGA)
  - 11421 LUTS, 12012 registers and 4 BRAMs

DUT drivers

- Hierarchical Linux Driver
  - ~1900 lines of C code
- Driver of SystemC model
  - ~1200 lines of C code

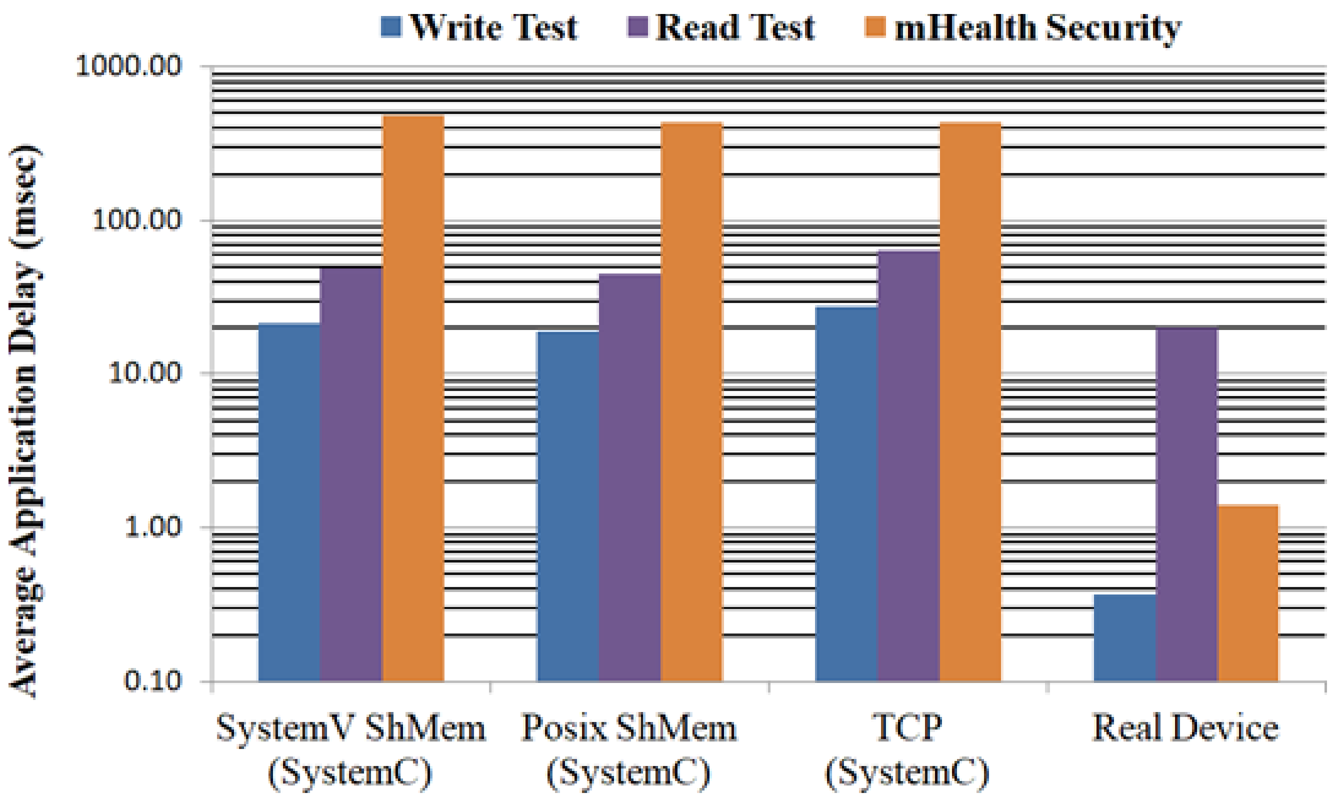
mHealth Testbench

- ~4500 lines of code

Read/Write Testbench

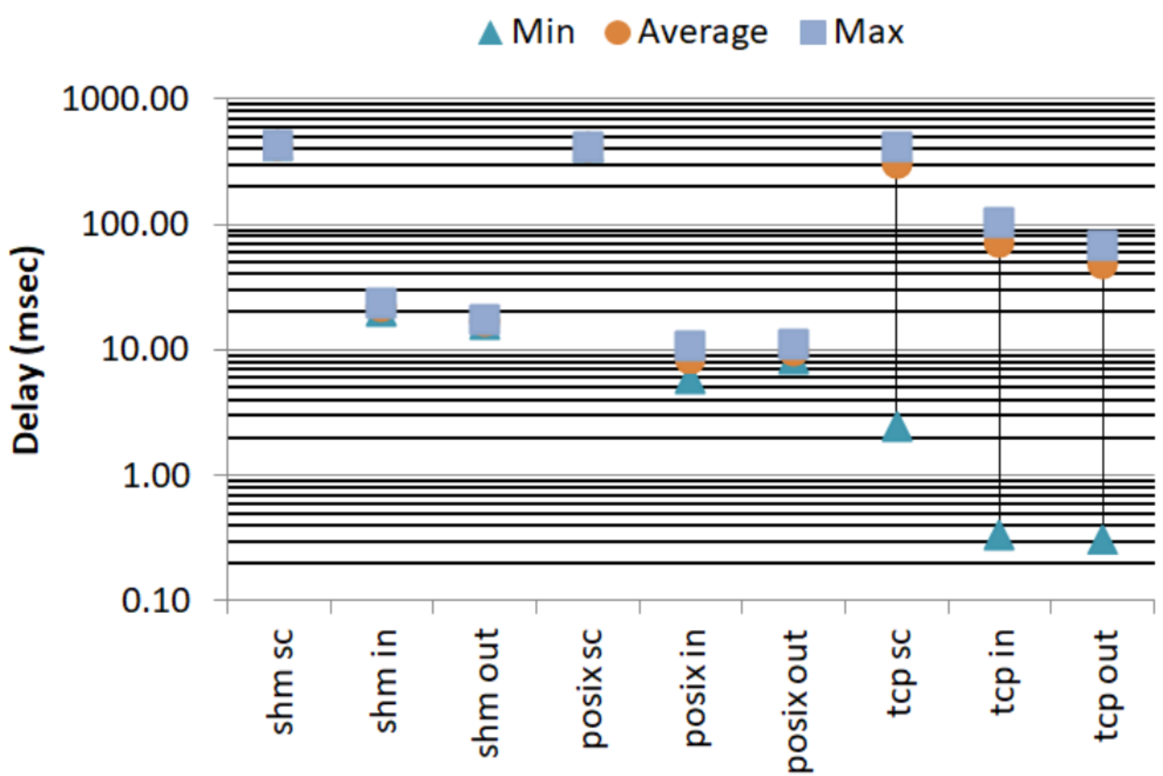
- ~500 lines of code each

Results - Average Application Delay



- POSIX shared memory is 6.5 to 11.1% faster than SystemV
- POSIX shared memory is 22 to 23% faster than TCP

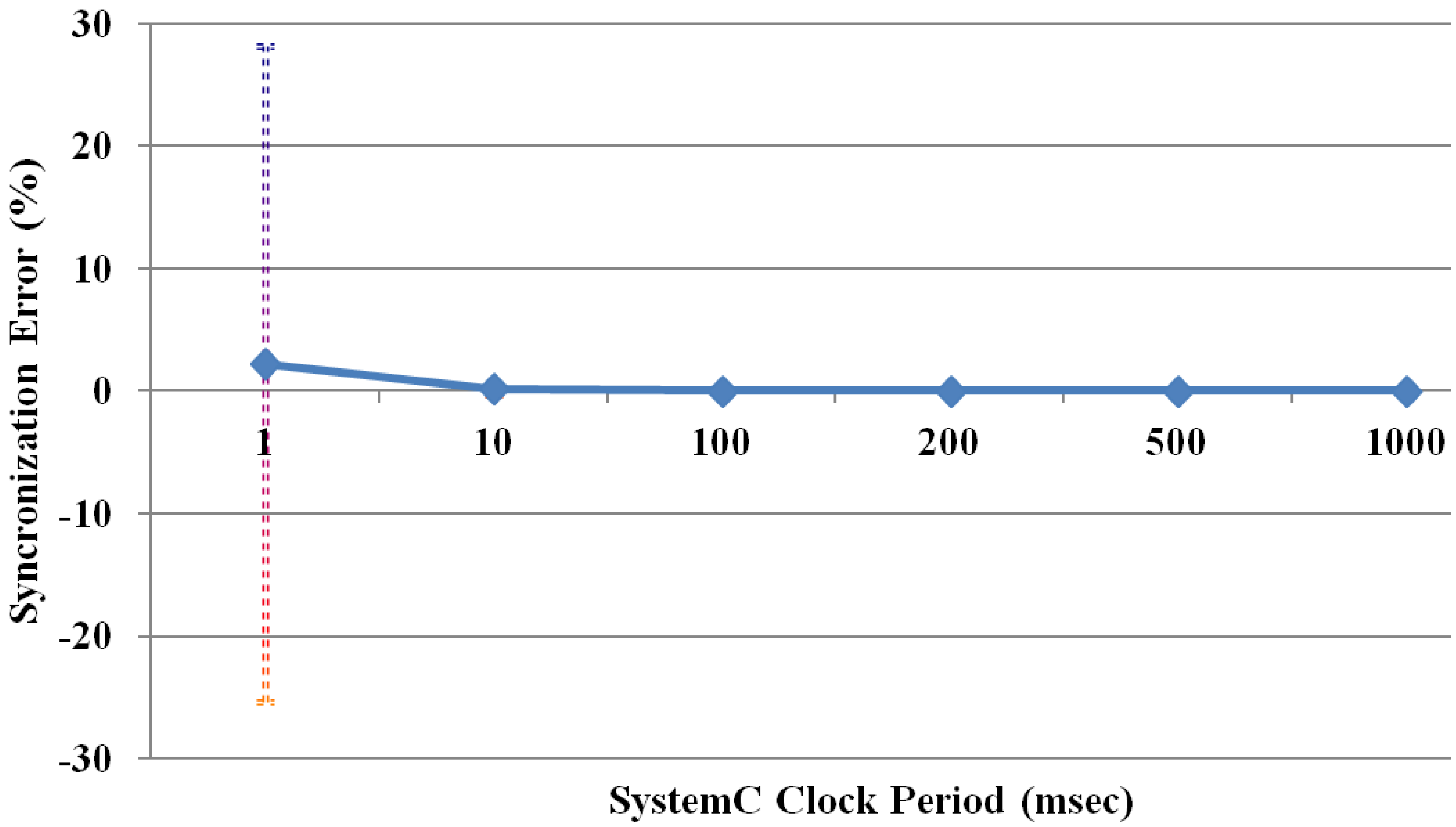
Results - Minimum, Average and Maximum Delay



- Asynchronous channel I/O delay compared to SystemC delay is
  - 5.2% for POSIX shared memory
  - 5.6% for SystemV shared memory
  - up to 41.0% for TCP socket (ignoring ethernet I/F delay)

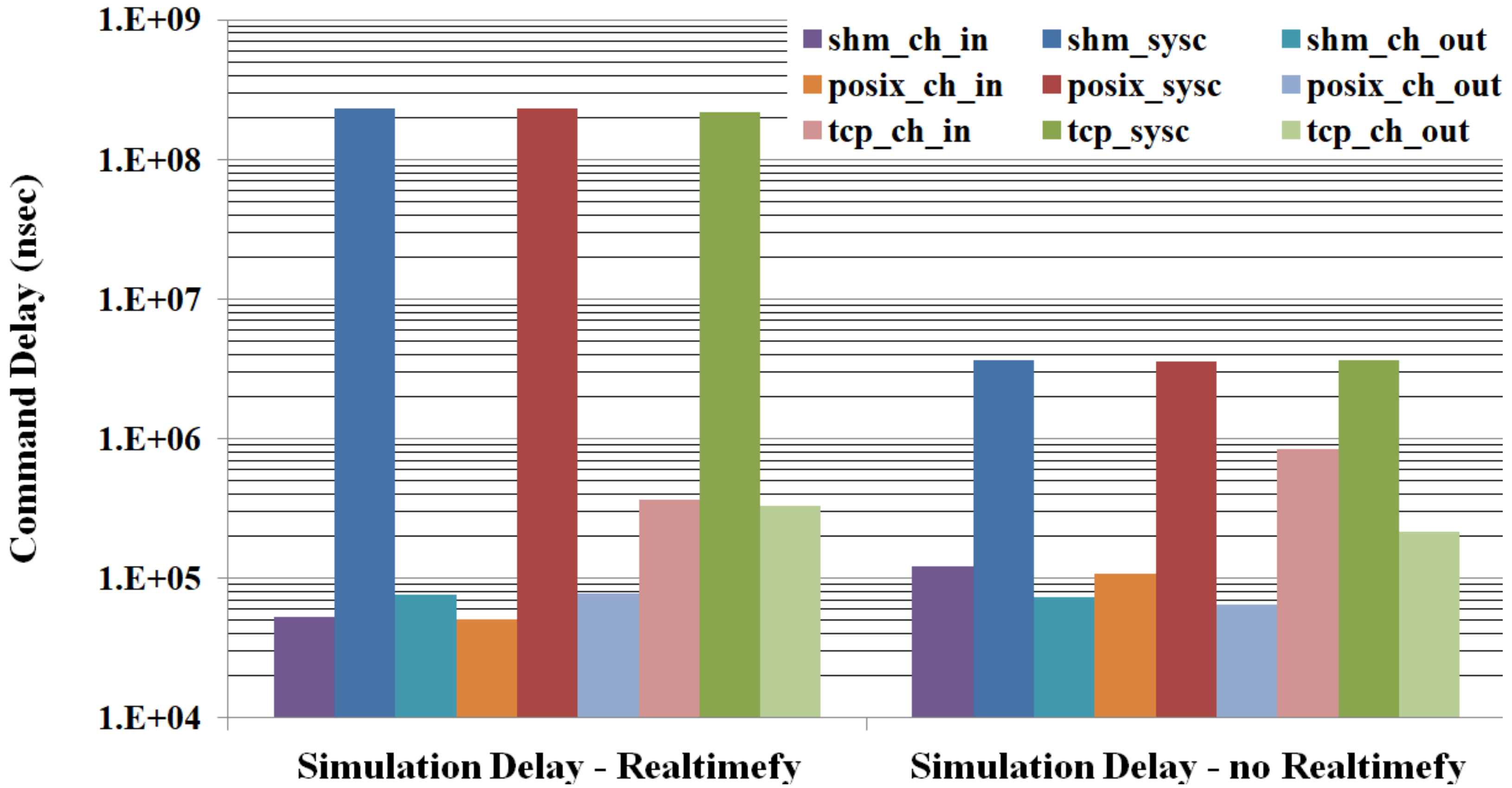
- *_sc values: SystemC model delay
- *_in values: channel input delay
- _out values: channel output delay

Results - Synchronization Error using Realtimify



- If SystemC period exceeds 1ms, the synchronization error is very small

Results - Simulation Delay



- Overhead differs by only one order of magnitude

Conclusion

Our open source, near real-time platform

- enables co-validation of HW DUT running on **a real system** with system-level IP
- implements a non-intrusive asynchronous channel for shared memory or TCP socket communications with the application
  - POSIX shared memory model is more efficient in terms of execution delay, but
  - SystemC model can also be run on a remote (large-scale) system using TCP

Code (GNU GPL v2) available from:

- https://github.com/apapagrigoriou/SC_CoSiM (cosimulation platform)
- https://github.com/angmouzakitis/student_xohw18-187 (testbench)

Future Work

- Drivers of real world applications could be extended to support code execution (and not just memory access over AXI)
- The asynchronous channel and System-Level IP could be extended to react to runtime events from external devices
- The HW DUT could be embedded in a platform that uses a real-time OS (e.g. ThreadX) to examine real-time co-simulation aspects

Questions ?

Additional Channel Methods

- *void nb_push (async_packet Packet);*
- *bool can_push (void);*
- *void pull (async_packet &Packet);*
- *bool can_pull (void);*
- *void put (async_packet Packet);*
- *bool can_put (void)*
- *void get (async_packet &Packet);*
- *bool can_get (void)*