Combining Model Checking and Testing in a Continuous HW/SW Co-Verification Process

Paula Herber, Florian Friedemann, and Sabine Glesner

Berlin Institute of Technology
Software Engineering for Embedded Systems Group

TAP - Tests and Proofs

Zurich, July 2009

Motivation

HW/SW Co-Design

- modeling and simulation with *system level design languages*
- stepwise *refinement* from abstract design to implementation
- SystemC
  - designs are *executable* on different abstraction levels
  - validation and verification by **co-simulation**

Problems

- impossible to cover all possible input scenarios (*incomplete*)
- consistency between abstraction levels hard to ensure
- limited degree of automatization (*manual evaluation*)

Motivation

HW/SW Co-Design

- modeling and simulation with *system level design languages*
- stepwise *refinement* from abstract design to implementation
- SystemC
  - designs are *executable* on different abstraction levels
  - validation and verification by **co-simulation**

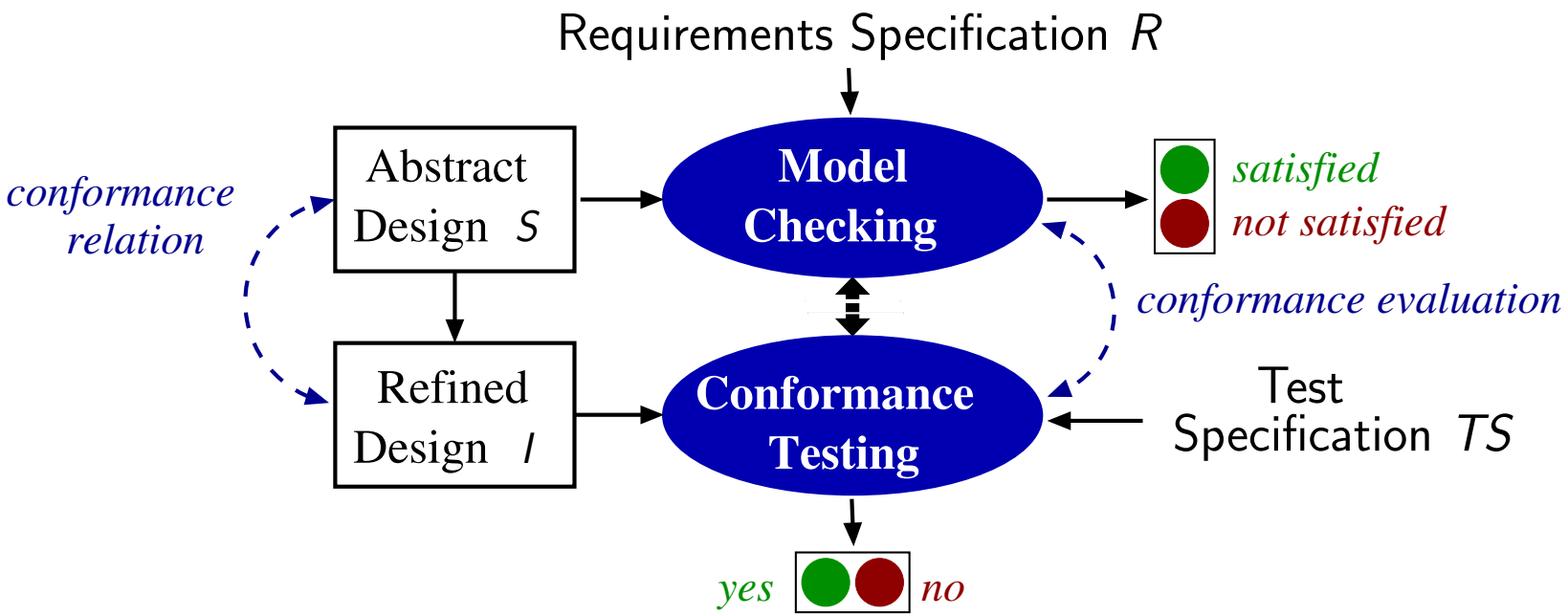
Problems

- impossible to cover all possible input scenarios (*incomplete*)
- consistency between abstraction levels hard to ensure
- limited degree of automatization (*manual evaluation*)

How can we assure quality in a more systematic way?

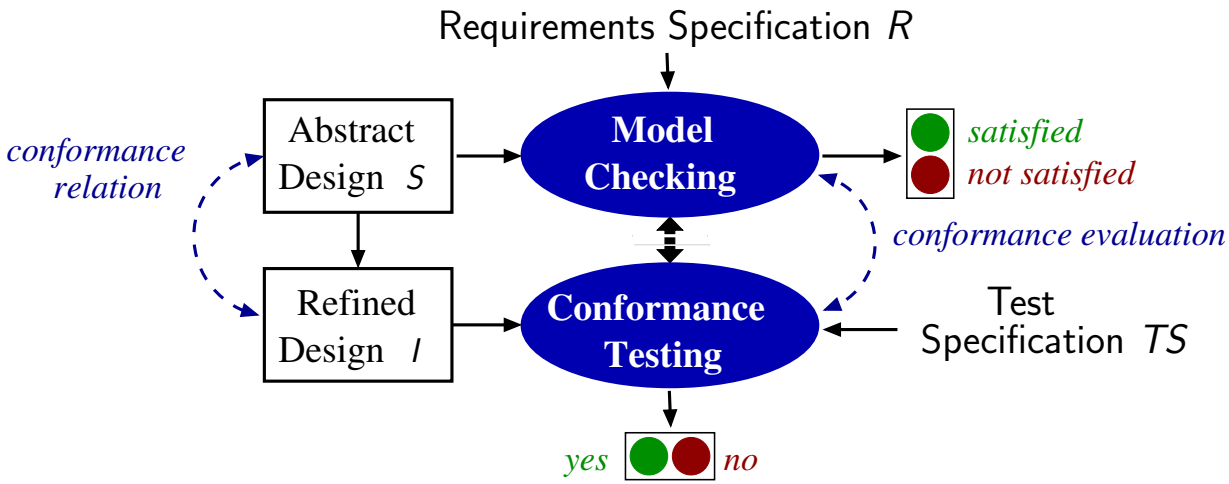
Continuous HW/SW Co-Verification Approach

1. verify requirements on abstract design via model checking
2. generate conformance tests for each refined design



Continuous HW/SW Co-Verification Approach

1. verify requirements on abstract design via model checking
2. generate conformance tests for each refined design



↯ **but**: semantics of SystemC is only **informally** defined

➠ map SystemC to UPPAAL timed automata [CODES+ISSS 2008]

➠ **use the UPPAAL model to generate conformance tests!**

SystemC

- introduced by the Open SystemC Initiative (OSCI) 1999
- semantics is (informally) defined in IEEE Std. 1666-2005

SystemC as system level design language

- description of both hardware and software on different levels of abstraction
- extends C++ by concurrency, time, hardware data types, reactivity, hierarchy, and abstract communication

SystemC as framework for HW/SW co-simulation

- light-weight simulation kernel executes SystemC designs in a discrete-event simulation

Uppaal

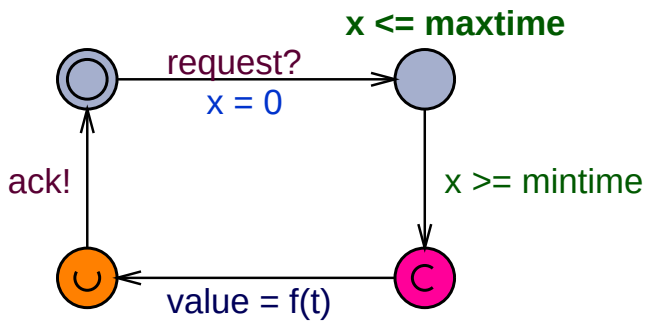
- modeling, simulation, and verification of timed automata
- jointly developed by the Universities of UPPsala and AALborg

Timed automata

- finite-state machines extended by **clock variables**
- **clock constraints** model time-dependent behavior
- Networks of timed automata model concurrent processes

Extensions in Uppaal

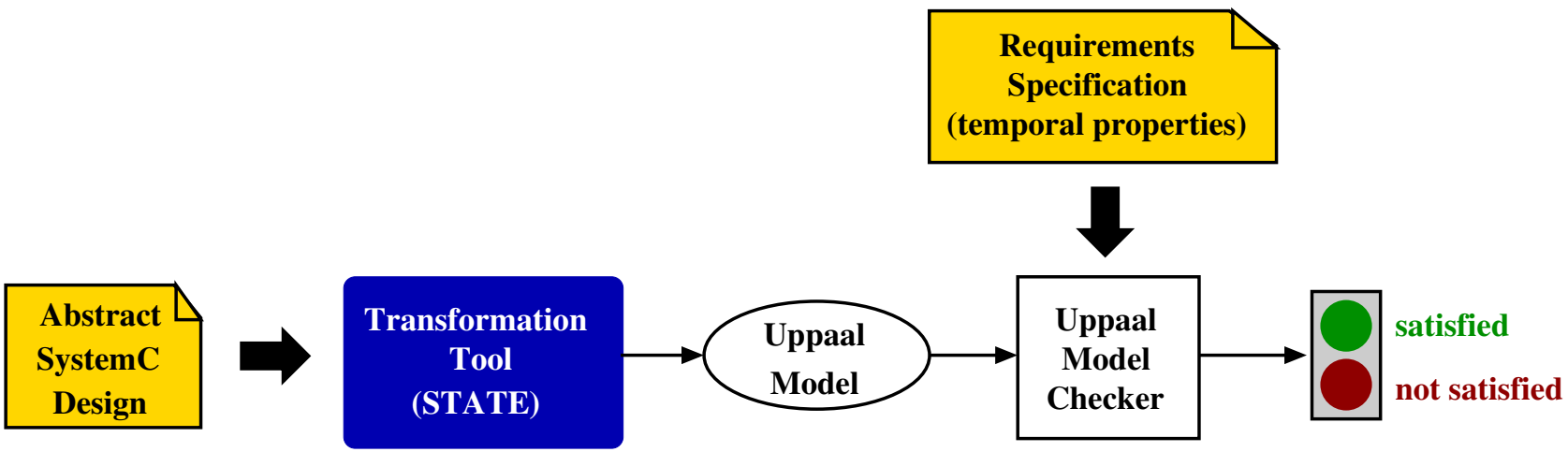
- parameterized timed automata templates
- **data variables** with bounded domains
- binary and broadcast channels
- **urgent** and **committed** locations



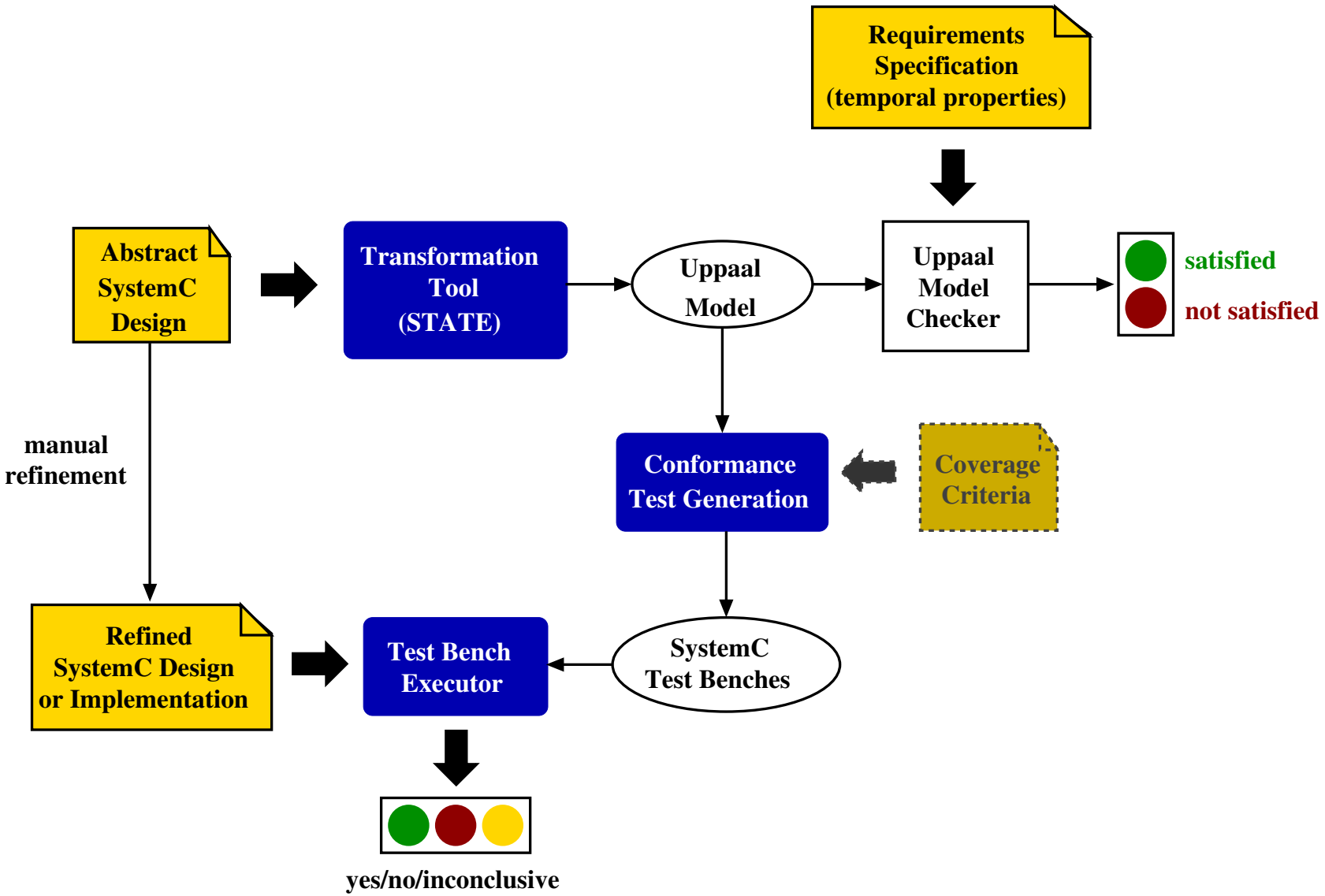
Model Checking of SystemC Designs

[Herber, Fellmuth, Glesner, CODES+ISSS 2008]

1. transform SystemC designs into UPPAAL timed automata
2. use the UPPAAL model checker to prove safety, liveness, and timing properties



HW/SW Co-Verification Framework



Test Model

- embedded systems interact closely with their environment
- simulation requires generation of inputs and consumption of outputs, this is provided by
  - a **test bench** in SystemC
  - an explicit environment or **test model** in UPPAAL

SystemC test bench

1. input generator:
   provides an input trace
2. output monitor:
   accepts all possible outputs

translation to UPPAAL

1. → *test automaton*
2. → *generic tester*

} **test model**

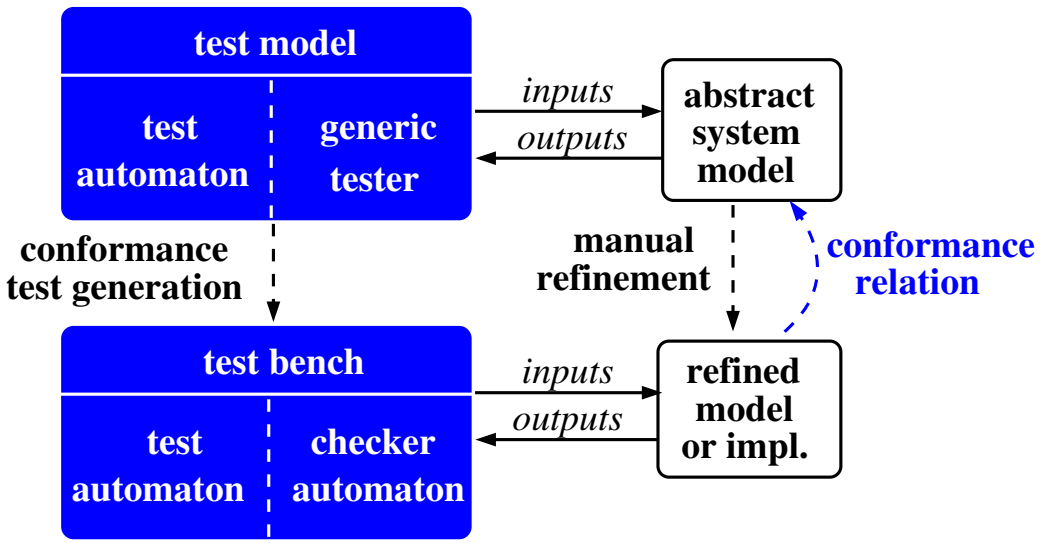
Conformance Test Generation

Goal:

- compute all possible outputs of a given UPPAAL model

Approach:

- execute test model together with abstract system model
- record traces in the *generic tester*
- construct a *checker automaton* from that



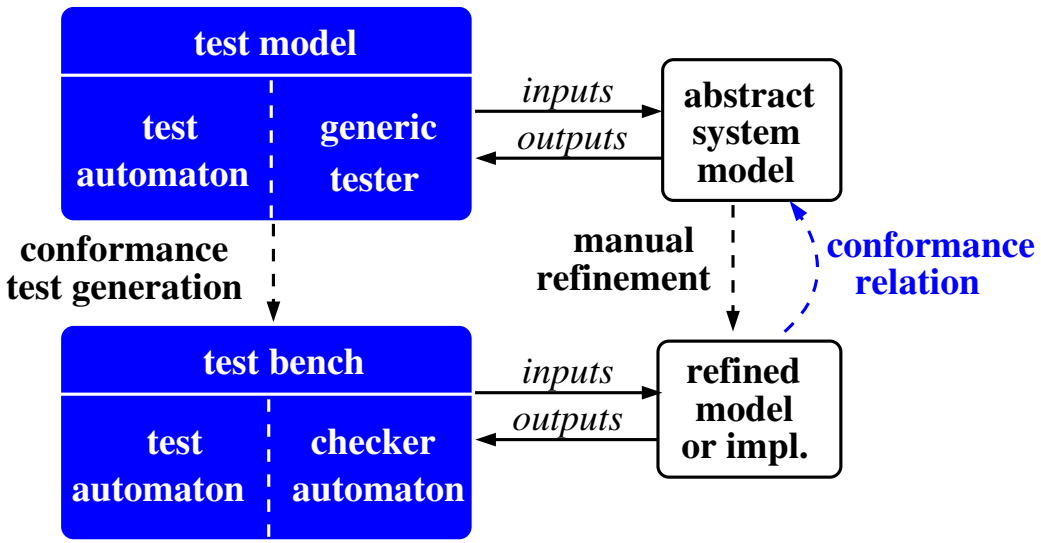
Conformance Test Generation

Goal:

- compute all possible outputs of a given UPPAAL model

Approach:

- execute test model together with abstract system model
- record traces in the *generic tester*
- construct a *checker automaton* from that



➠ **requires symbolic execution of the Uppaal model**

Symbolic Execution

Symbolic Semantics of Uppaal:

- uses *clock zones* to abstract from time points
- symbolic state: location vector, clock zone, variable valuations

Symbolic Execution:

1. start with the initial symbolic state
2. compute all possible symbolic successor states

Challenge:

- compute outputs *offline* for *non-deterministic* specifications

➠ restrict to finite input traces

➠ identify and aggregate semantically equivalent symbolic states

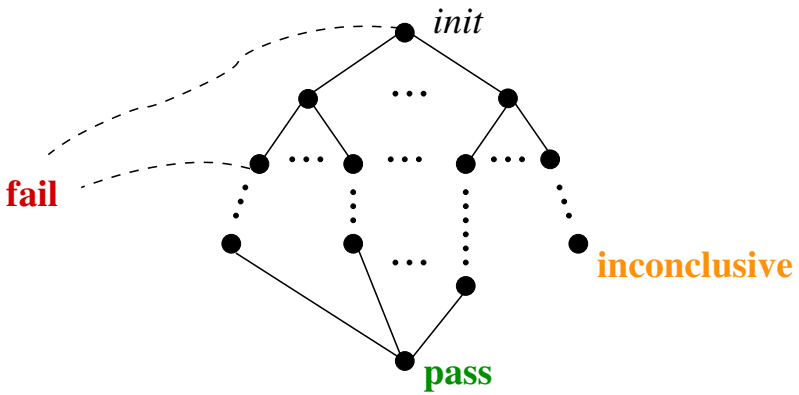
➠ limit the number of internal computation steps

Checker Automaton

- result of symbolic execution: all possible output traces

Construction of a checker automaton

1. merge end nodes into a **pass** node
2. mark nodes with **inconclusive** if computation step limit exceeded
3. each unexpected trace leads to the test verdict **fail**

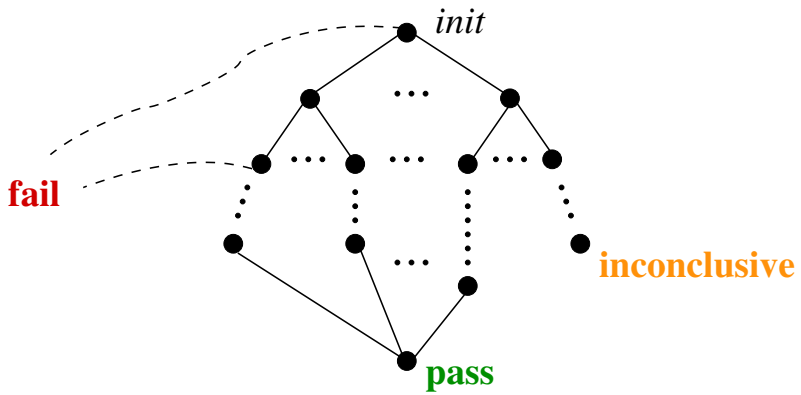


Checker Automaton

- result of symbolic execution: all possible output traces

Construction of a checker automaton

1. merge end nodes into a **pass** node
2. mark nodes with **inconclusive** if computation step limit exceeded
3. each unexpected trace leads to the test verdict **fail**



➠ from checker automata, SystemC test benches for automatic conformance evaluation can be generated automatically (*tbd*)

Case Study: Packet Switch Example

1. Transformation from SystemC to UPPAAL and model checking of safety and timing properties

	Computation time (in seconds)			
	1m1s	2m1s	1m2s	2m2s
transformation	1.46	1.59	1.52	1.70
no deadlock	20.82	54.90	42.21	209.56
forward within time limit	127.70	45.04	296.89	543.18

2. Translation into an executable representation and conformance test generation

	Computation time (in seconds)			
	1m1s	2m1s	1m2s	2m2s
translation	7.82	9.91	9.38	10.93
compilation	8.54	10.04	9.05	9.7
test generation	8.75	14.32	23.95	34.14

Conclusion

Continuous HW/SW co-verification process

- ✓ combines the power of model checking and of testing
- ✓ yields an *automated co-verification flow* for SystemC designs

Conformance test generation

- ✓ *offline* for *non-deterministic* specifications
- ✓ allows for *automated testing* whether a refined SystemC design conforms to a *verified* abstract SystemC design

Conclusion

Continuous HW/SW co-verification process

- ✓ combines the power of model checking and of testing
- ✓ yields an *automated co-verification flow* for SystemC designs

Conformance test generation

- ✓ *offline* for *non-deterministic* specifications
- ✓ allows for *automated testing* whether a refined SystemC design conforms to a *verified* abstract SystemC design

Future Work:

- generate SystemC test benches from checker automata
- evaluate error detecting capability by a larger case study
- coverage-driven input selection