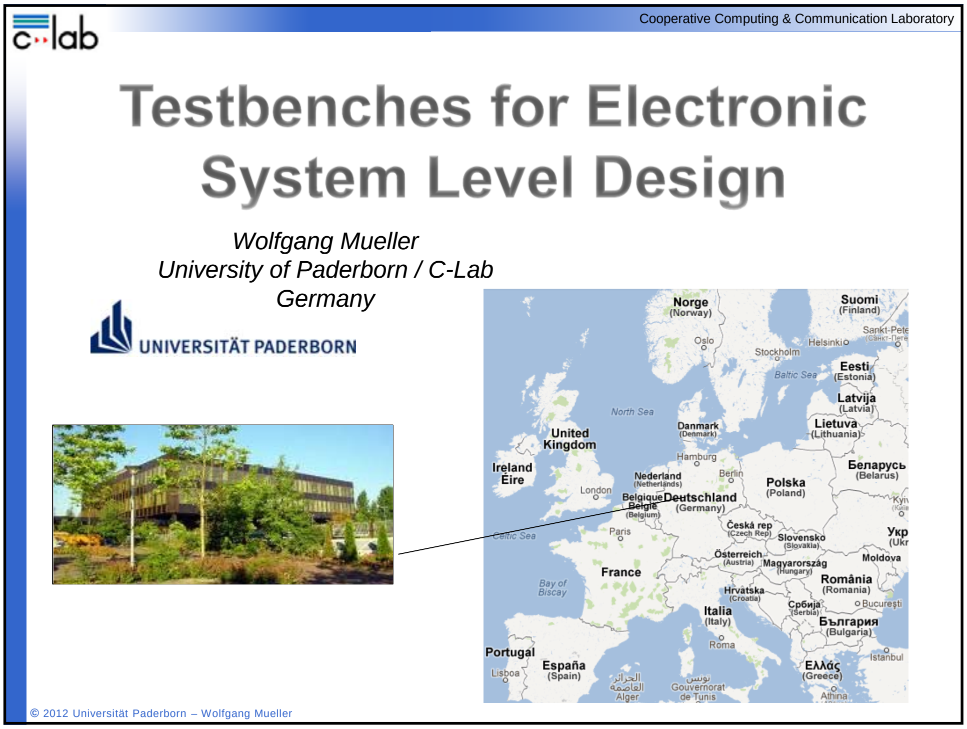
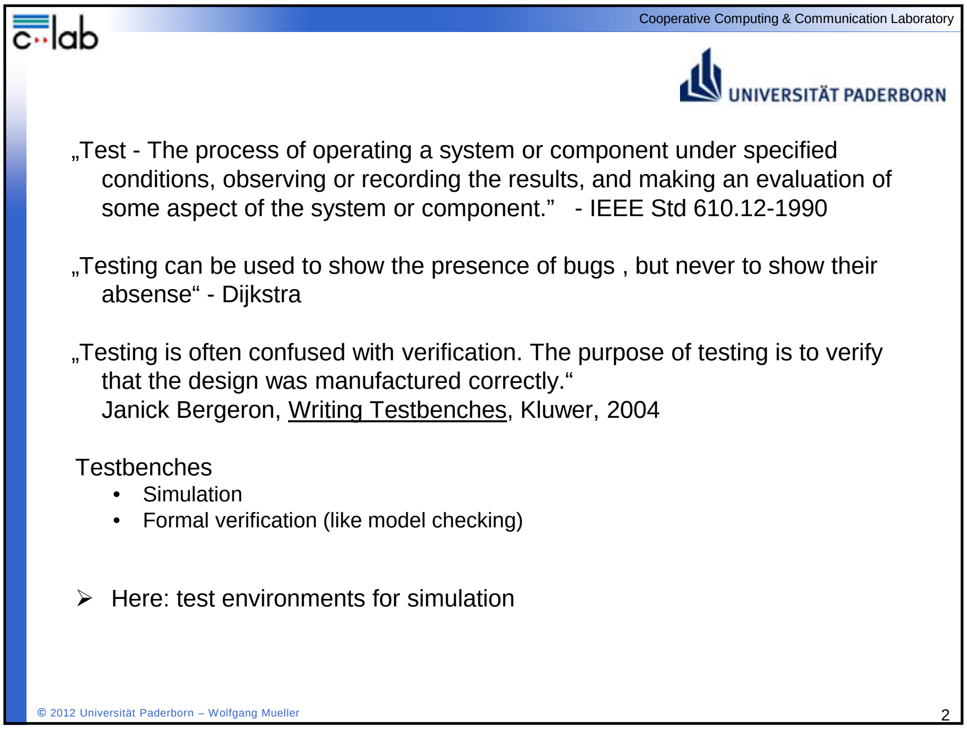


Cooperative Computing & Communication Laboratory

Testbenches for Electronic System Level Design

Wolfgang Mueller
University of Paderborn / C-Lab
Germany



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

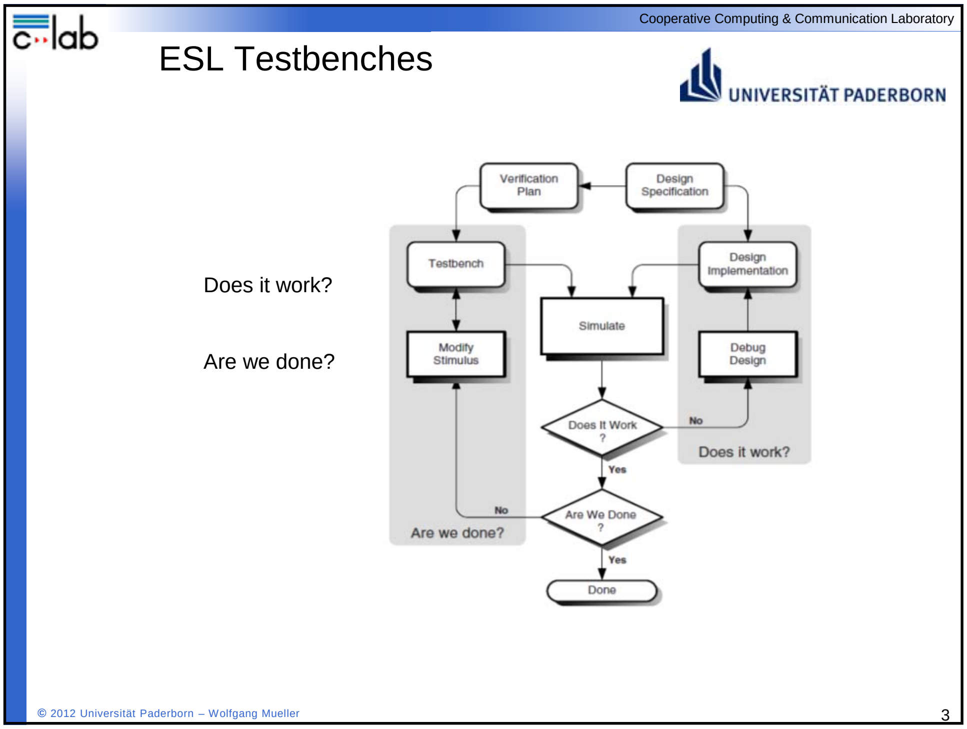
„Test - The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.” - IEEE Std 610.12-1990

„Testing can be used to show the presence of bugs , but never to show their absense“ - Dijkstra

„Testing is often confused with verification. The purpose of testing is to verify that the design was manufactured correctly.“
Janick Bergeron, Writing Testbenches, Kluwer, 2004

Testbenches
- Simulation
- Formal verification (like model checking)

➢ Here: test environments for simulation



Cooperative Computing & Communication Laboratory

ESL Testbenches

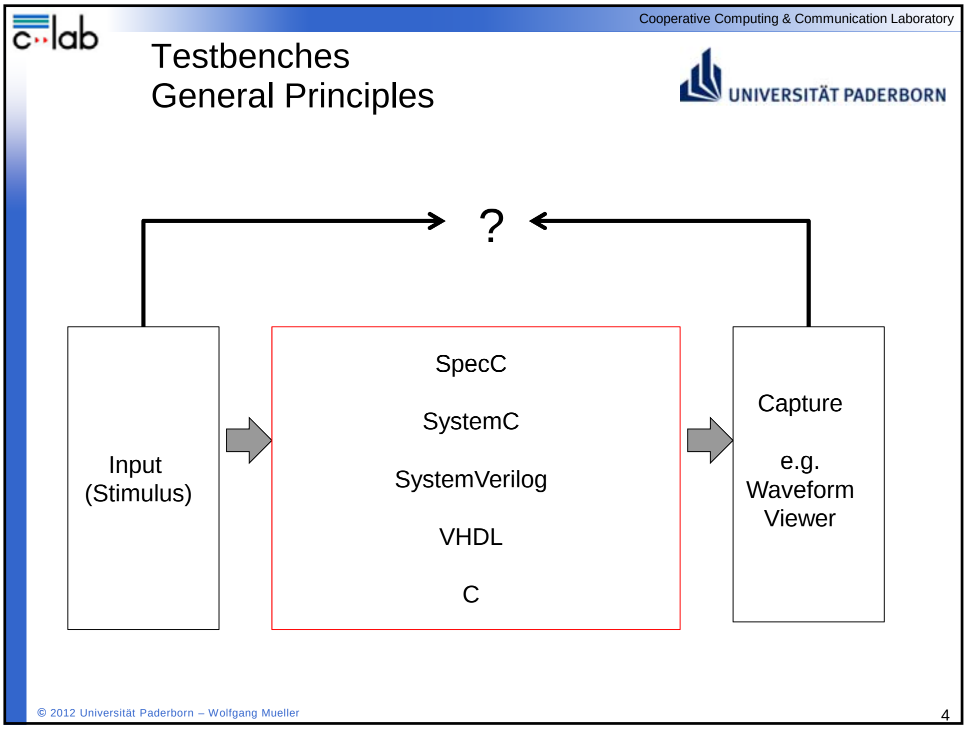
UNIVERSITÄT PADERBORN

Does it work?

Are we done?

© 2012 Universität Paderborn – Wolfgang Mueller

3



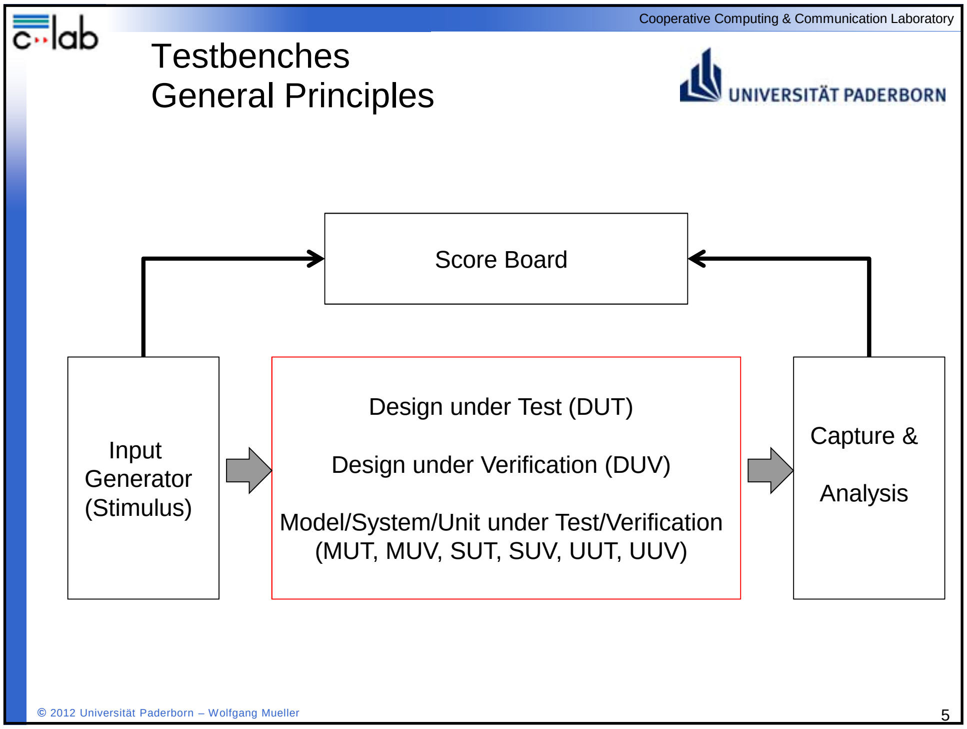
Cooperative Computing & Communication Laboratory

Testbenches
General Principles

UNIVERSITÄT PADERBORN

© 2012 Universität Paderborn – Wolfgang Mueller

4



Cooperative Computing & Communication Laboratory

Testbenches General Principles

UNIVERSITÄT PADERBORN

Score Board

Input Generator (Stimulus)

Design under Test (DUT)

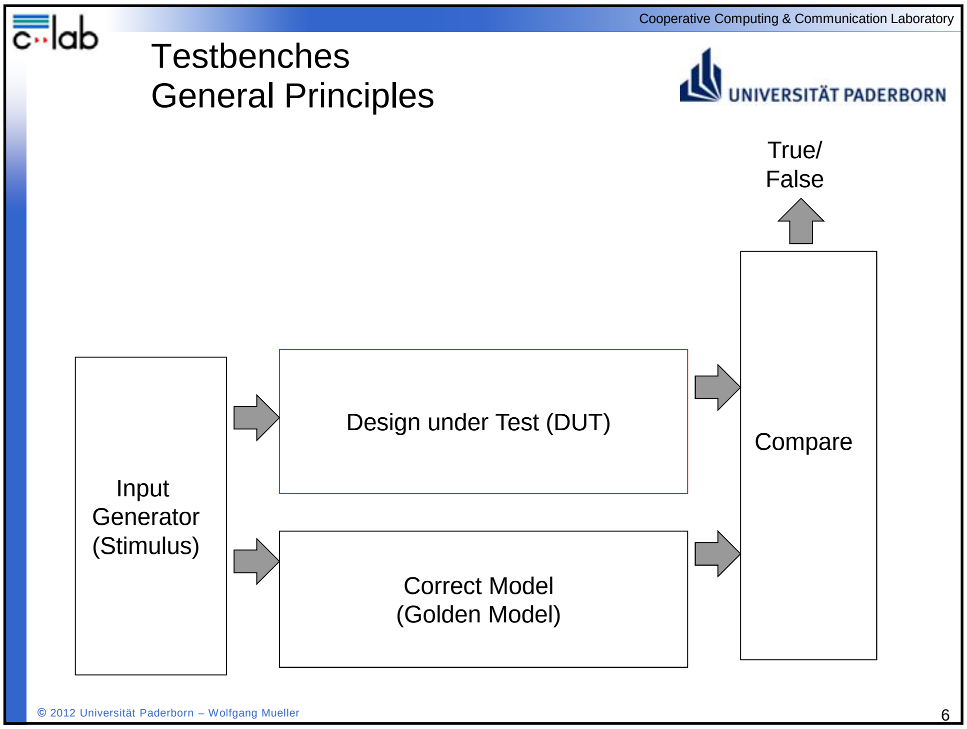
Design under Verification (DUV)

Model/System/Unit under Test/Verification (MUT, MUV, SUT, SUV, UUT, UUV)

Capture & Analysis

© 2012 Universität Paderborn – Wolfgang Mueller

5



Cooperative Computing & Communication Laboratory

Testbenches General Principles

UNIVERSITÄT PADERBORN

True/ False

Input Generator (Stimulus)

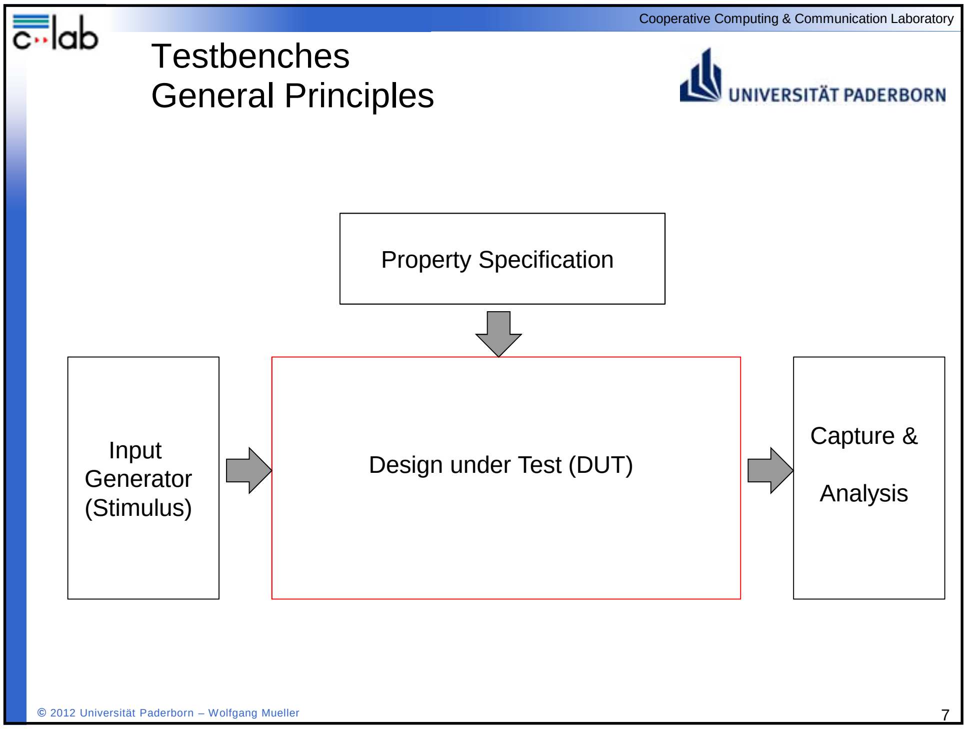
Design under Test (DUT)

Correct Model (Golden Model)

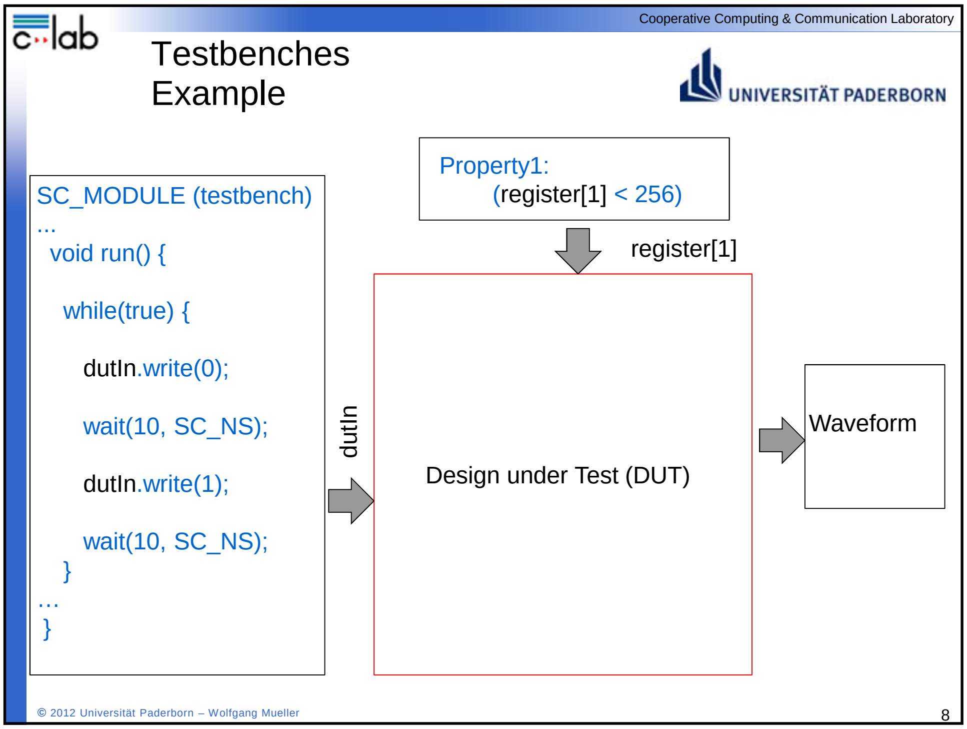
Compare

© 2012 Universität Paderborn – Wolfgang Mueller

6



Testbenches
General Principles



Testbenches
Example

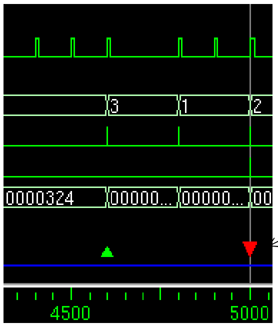
```
SC_MODULE (testbench)
...
  void run() {
    while(true) {
       dutIn.write(0);
       wait(10, SC_NS);
       dutIn.write(1);
       wait(10, SC_NS);
    }
...
}
```

```
Property1:
        (register[1] < 256)
```

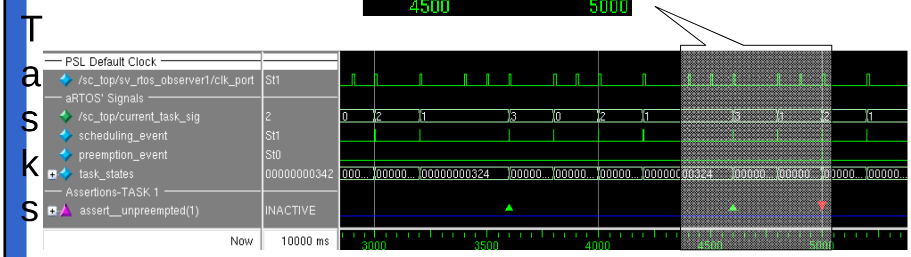
s for RT Tasks

Waveform example

QuestaSim ™
MentorGraphics

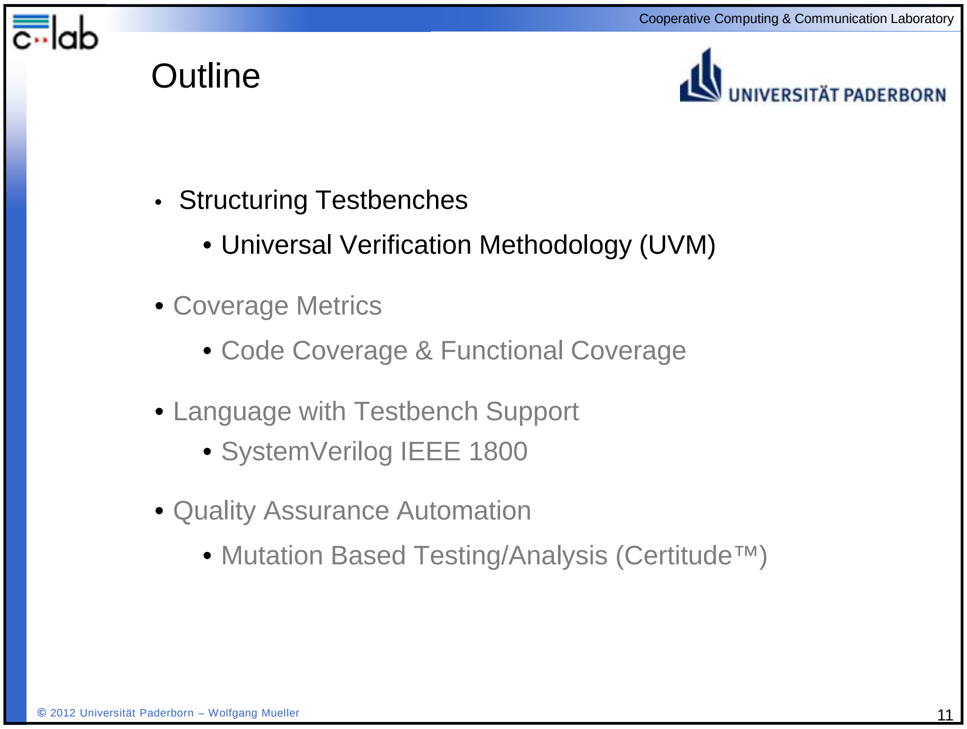


property failed


PSL Default Clock
/sc_top/sv_rtos_observer1/clk_port St1
aRTOS' Signals
/sc_top/current_task_sig 2
scheduling_event St1
preemption_event St0
task_states 00000000342
Assertions-TASK 1
assert__unpreempted(1) INACTIVE
Now 10000 ms

Outline

- Structuring Testbenches
  - Universal Verification Methodology (UVM)
- Coverage Metrics
  - Code Coverage & Functional Coverage
- Language with Testbench Support
  - SystemVerilog IEEE 1800
- Quality Assurance Automation
  - Mutation Based Testing/Analysis (Certitude™)

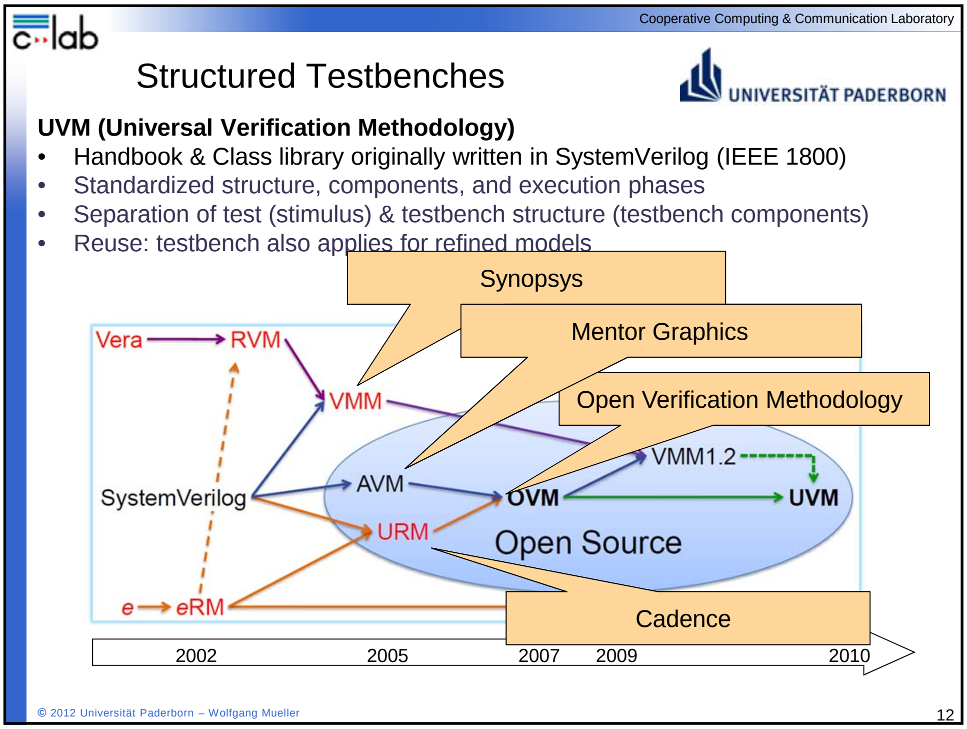


Cooperative Computing & Communication Laboratory

Outline

- Structuring Testbenches
  - Universal Verification Methodology (UVM)
- Coverage Metrics
  - Code Coverage & Functional Coverage
- Language with Testbench Support
  - SystemVerilog IEEE 1800
- Quality Assurance Automation
  - Mutation Based Testing/Analysis (Certitude™)

© 2012 Universität Paderborn – Wolfgang Mueller 11



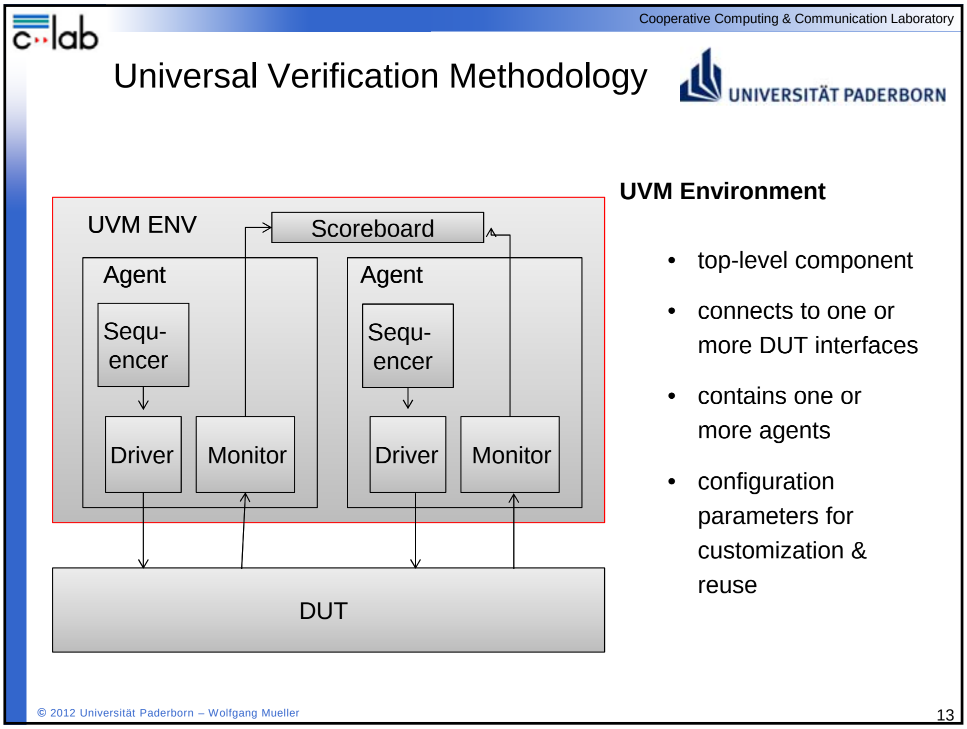
Cooperative Computing & Communication Laboratory

Structured Testbenches

UVM (Universal Verification Methodology)

- Handbook & Class library originally written in SystemVerilog (IEEE 1800)
- Standardized structure, components, and execution phases
- Separation of test (stimulus) & testbench structure (testbench components)
- Reuse: testbench also applies for refined models

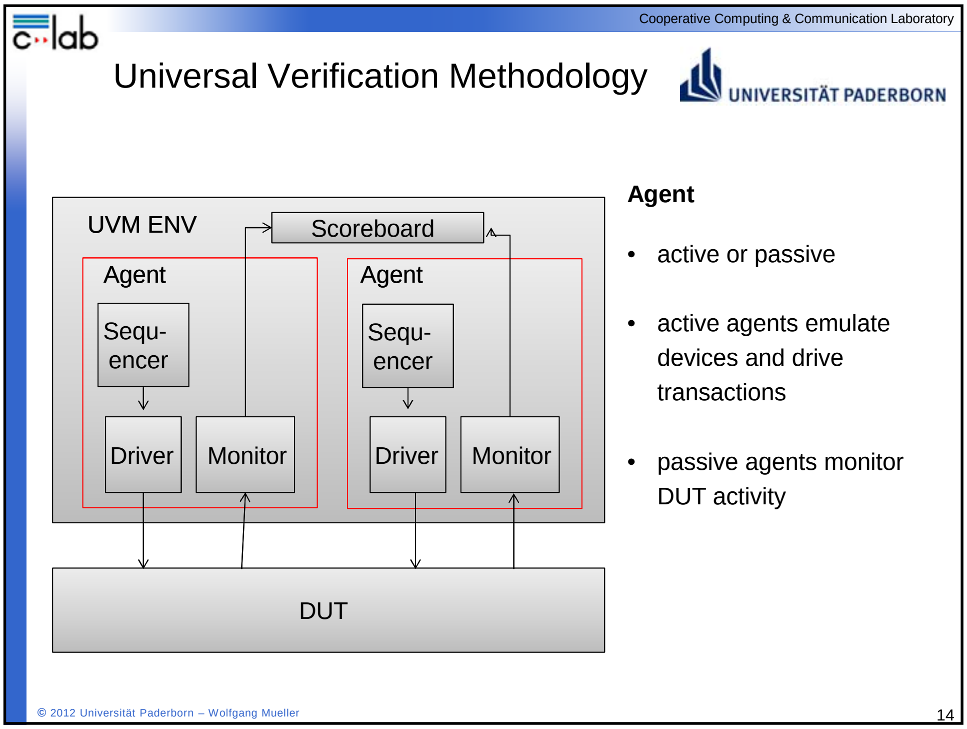
© 2012 Universität Paderborn – Wolfgang Mueller 12



Universal Verification Methodology

UVM Environment

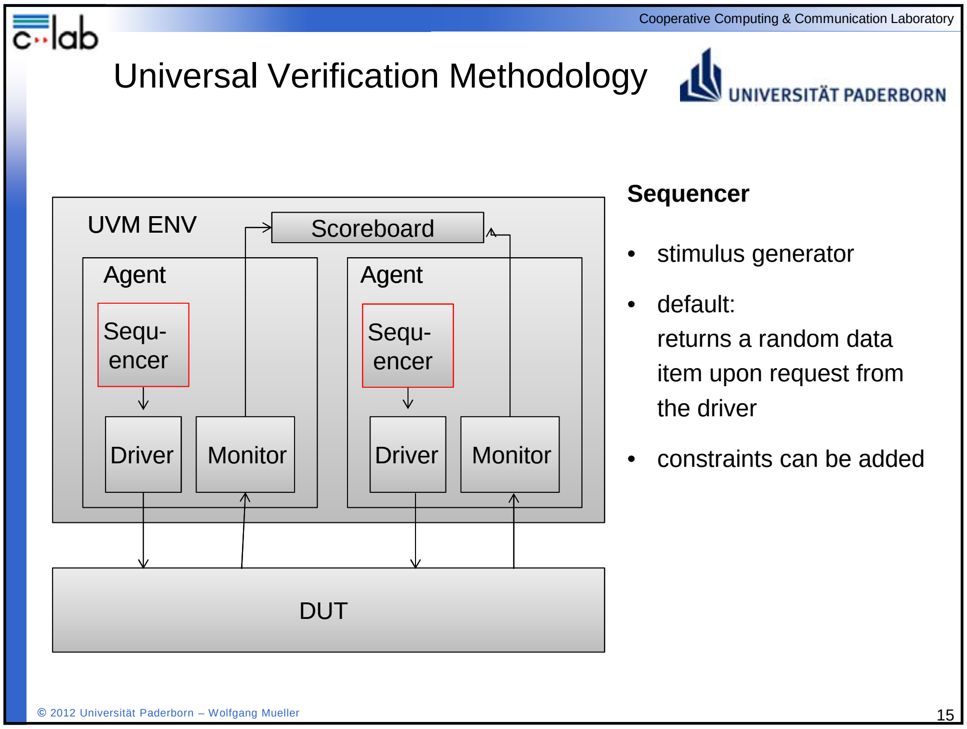
- top-level component
- connects to one or more DUT interfaces
- contains one or more agents
- configuration parameters for customization & reuse



Universal Verification Methodology

Agent

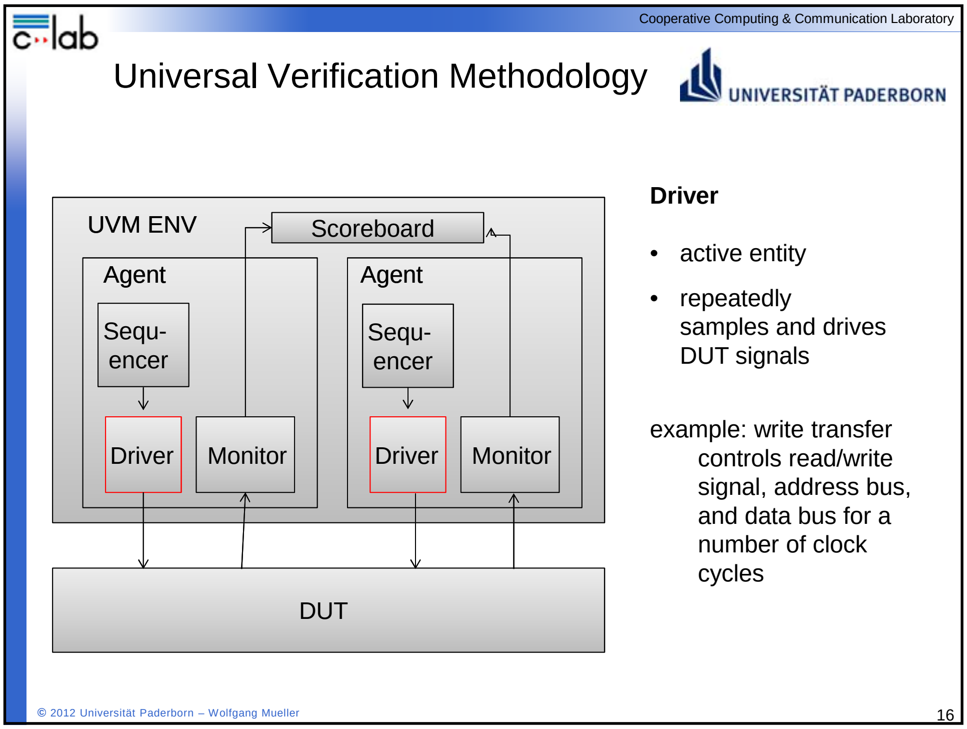
- active or passive
- active agents emulate devices and drive transactions
- passive agents monitor DUT activity



Universal Verification Methodology

Sequencer

- stimulus generator
- default:
  returns a random data item upon request from the driver
- constraints can be added

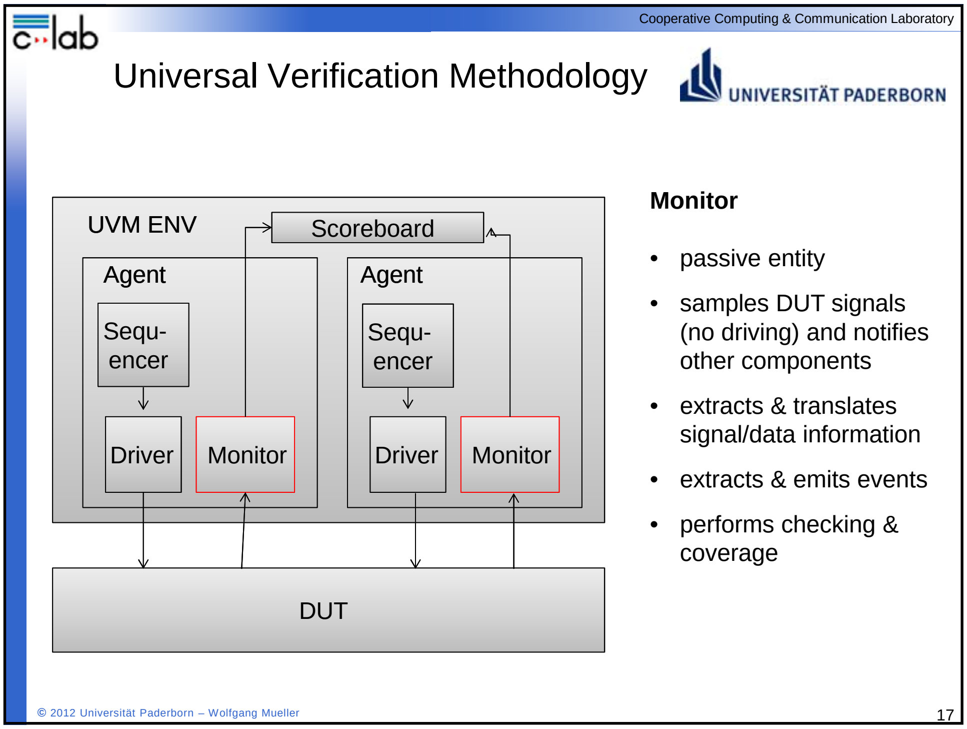


Universal Verification Methodology

Driver

- active entity
- repeatedly samples and drives DUT signals

example: write transfer
controls read/write signal, address bus, and data bus for a number of clock cycles



Cooperative Computing & Communication Laboratory

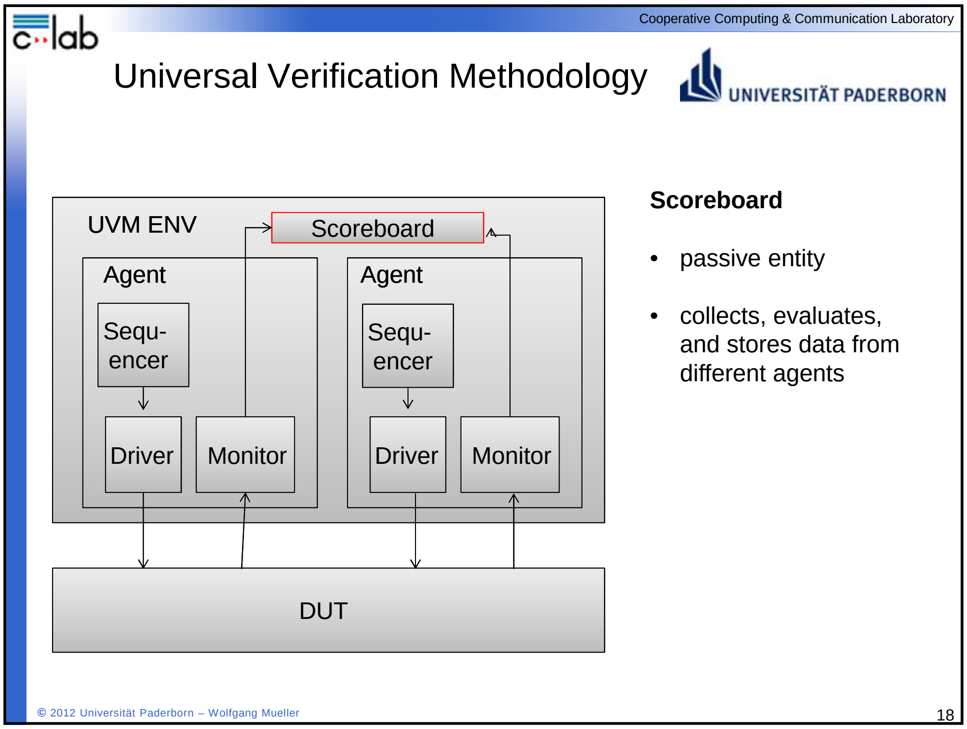
Universal Verification Methodology

UNIVERSITÄT PADERBORN

Monitor

- passive entity
- samples DUT signals (no driving) and notifies other components
- extracts & translates signal/data information
- extracts & emits events
- performs checking & coverage

17



Cooperative Computing & Communication Laboratory

Universal Verification Methodology

UNIVERSITÄT PADERBORN

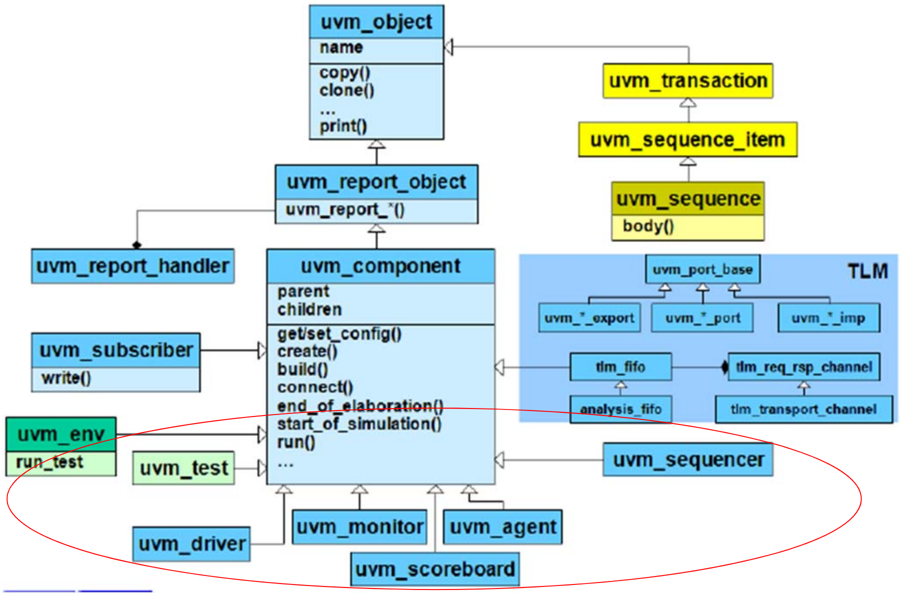
Scoreboard

- passive entity
- collects, evaluates, and stores data from different agents

18

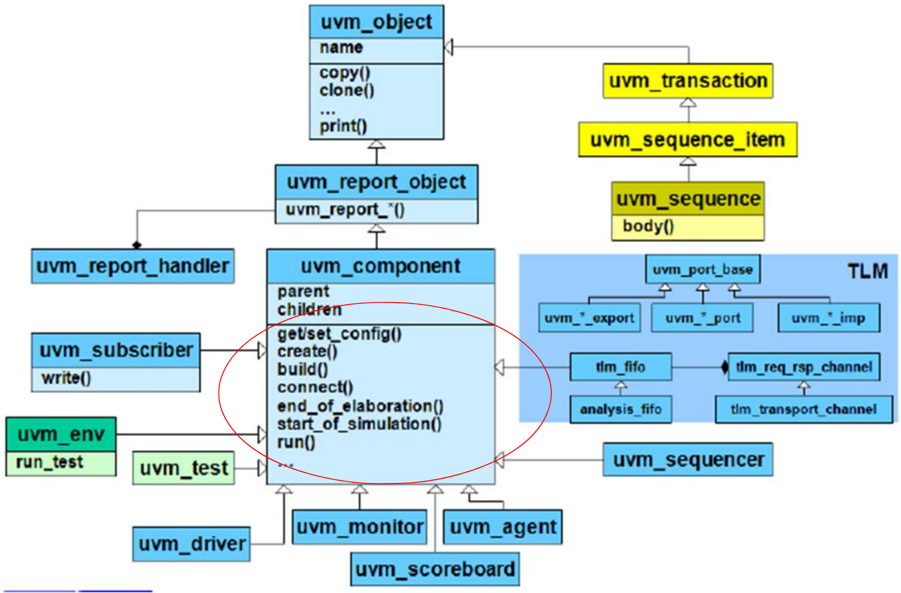
Universal Verification Methodology

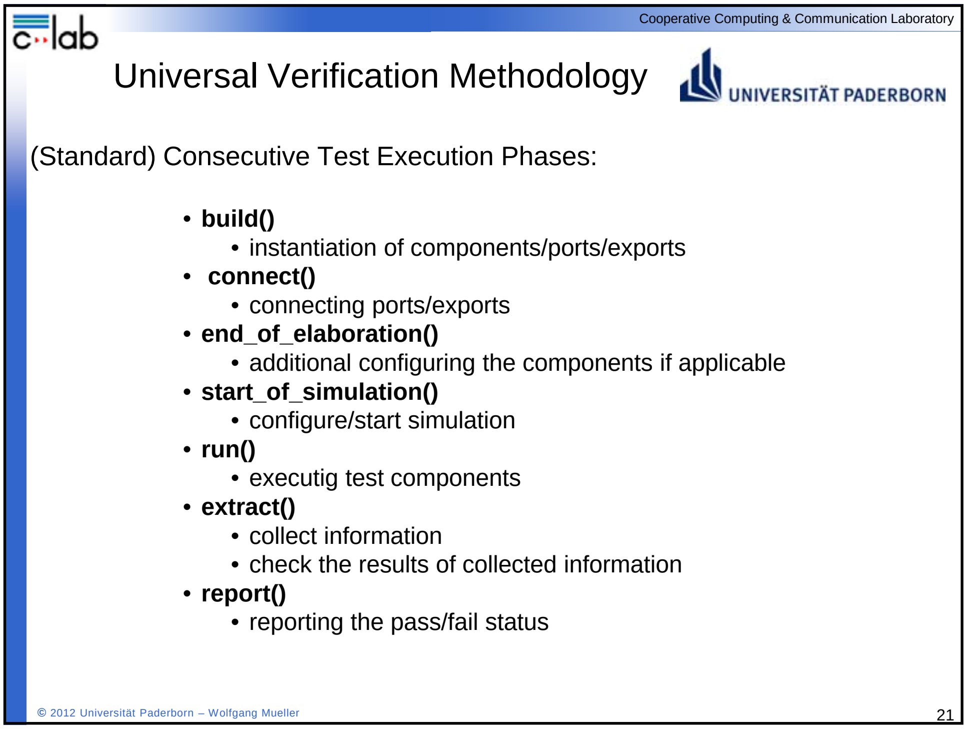
UNIVERSITÄT PADERBORN



Universal Verification Methodology

UNIVERSITÄT PADERBORN





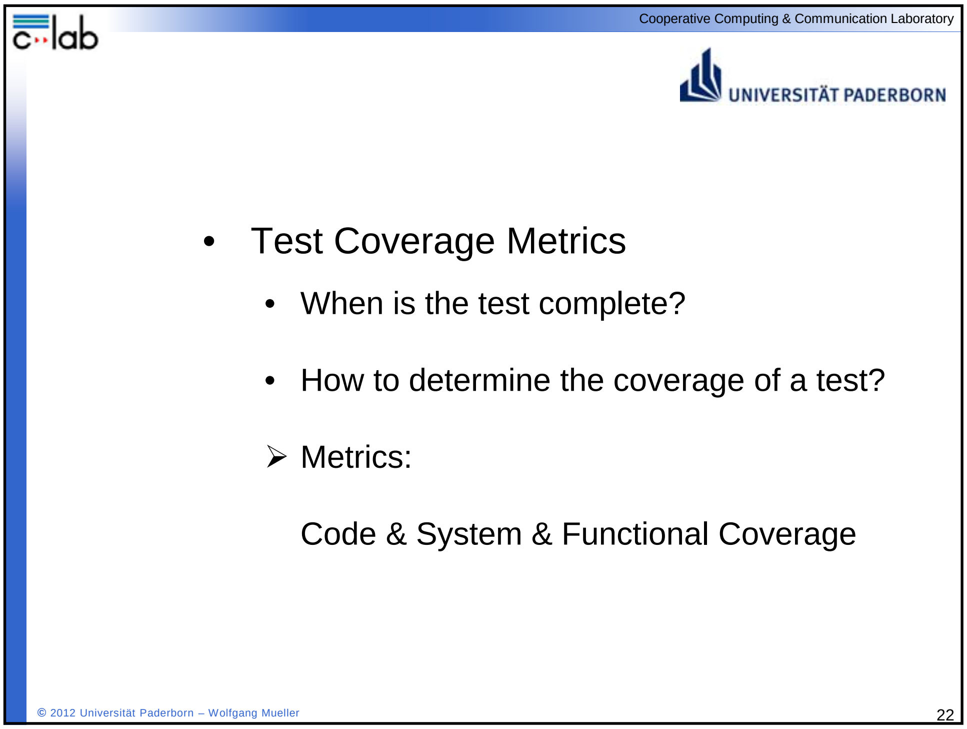
Cooperative Computing & Communication Laboratory

Universal Verification Methodology

(Standard) Consecutive Test Execution Phases:

- **build()**
  - instantiation of components/ports/exports
- **connect()**
  - connecting ports/exports
- **end_of_elaboration()**
  - additional configuring the components if applicable
- **start_of_simulation()**
  - configure/start simulation
- **run()**
  - executig test components
- **extract()**
  - collect information
  - check the results of collected information
- **report()**
  - reporting the pass/fail status

© 2012 Universität Paderborn – Wolfgang Mueller 21

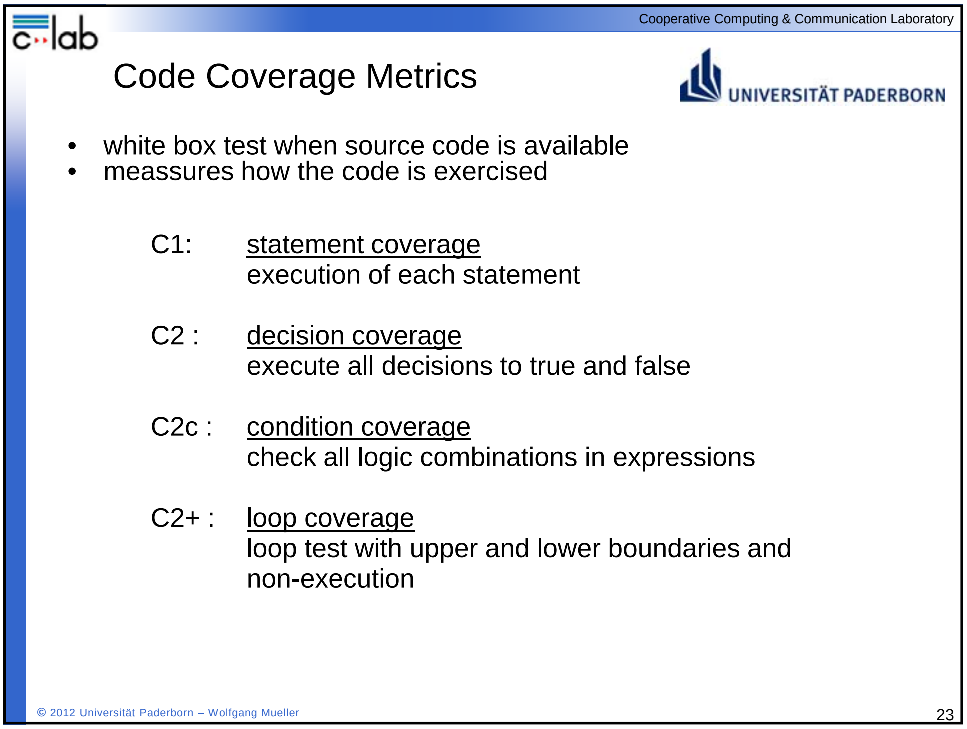


Cooperative Computing & Communication Laboratory

- Test Coverage Metrics
  - When is the test complete?
  - How to determine the coverage of a test?
  - ➢ Metrics:

    Code & System & Functional Coverage

© 2012 Universität Paderborn – Wolfgang Mueller 22



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

Code Coverage Metrics

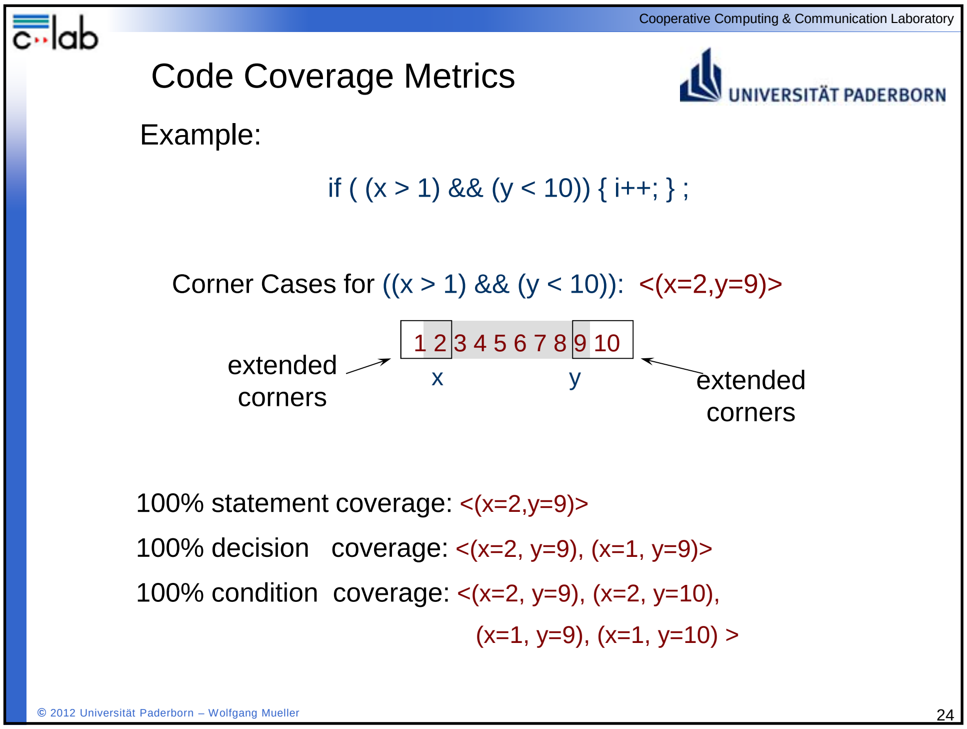
- white box test when source code is available
- meassures how the code is exercised

C1: statement coverage
execution of each statement

C2 : decision coverage
execute all decisions to true and false

C2c : condition coverage
check all logic combinations in expressions

C2+ : loop coverage
loop test with upper and lower boundaries and non-execution



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

Code Coverage Metrics

Example:

if ((x > 1) && (y < 10)) { i++; } ;

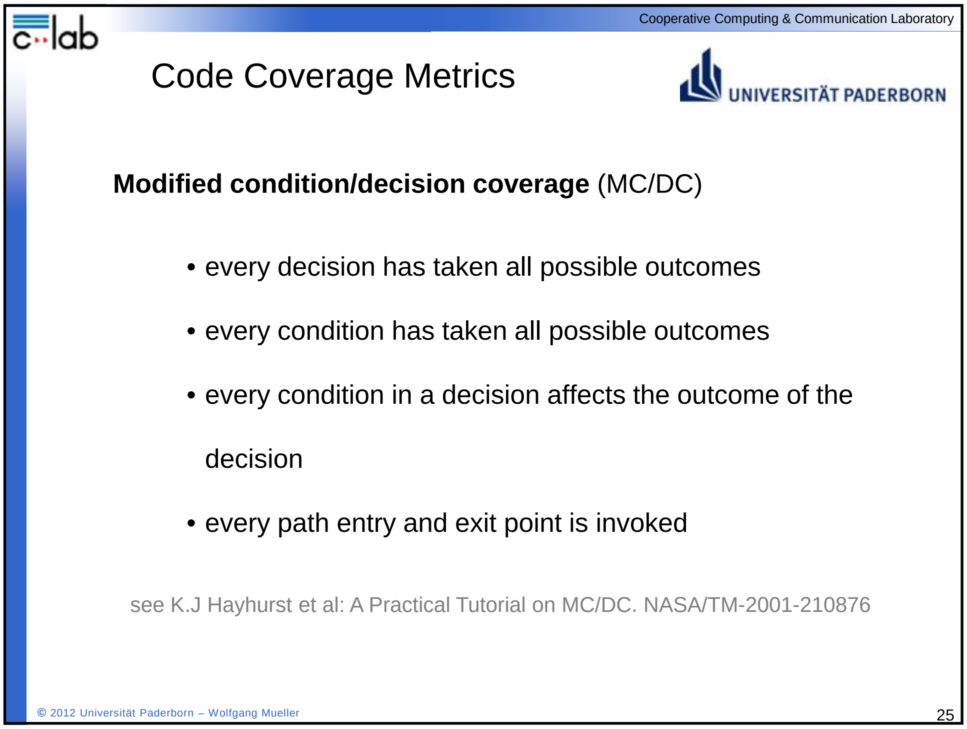
Corner Cases for ((x > 1) && (y < 10)): <(x=2,y=9)>

1 2 3 4 5 6 7 8 9 10
x
y
extended corners
extended corners

100% statement coverage: <(x=2,y=9)>

100% decision coverage: <(x=2, y=9), (x=1, y=9)>

100% condition coverage: <(x=2, y=9), (x=2, y=10),
(x=1, y=9), (x=1, y=10) >



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

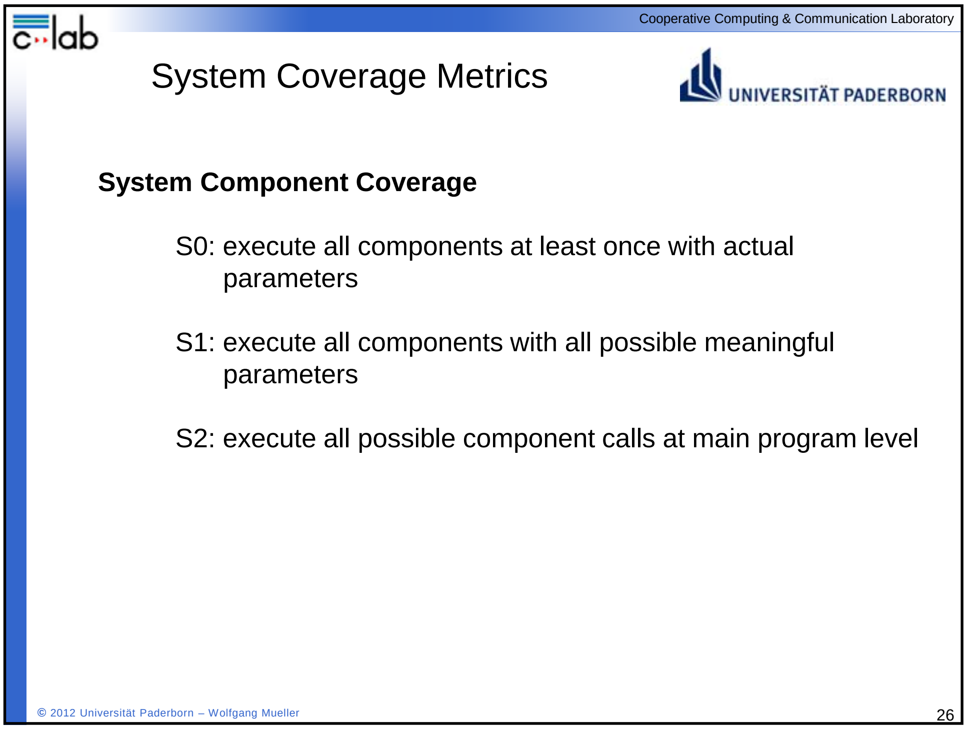
Code Coverage Metrics

Modified condition/decision coverage (MC/DC)

- every decision has taken all possible outcomes
- every condition has taken all possible outcomes
- every condition in a decision affects the outcome of the decision
- every path entry and exit point is invoked

see K.J Hayhurst et al: A Practical Tutorial on MC/DC. NASA/TM-2001-210876

© 2012 Universität Paderborn – Wolfgang Mueller 25



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

System Coverage Metrics

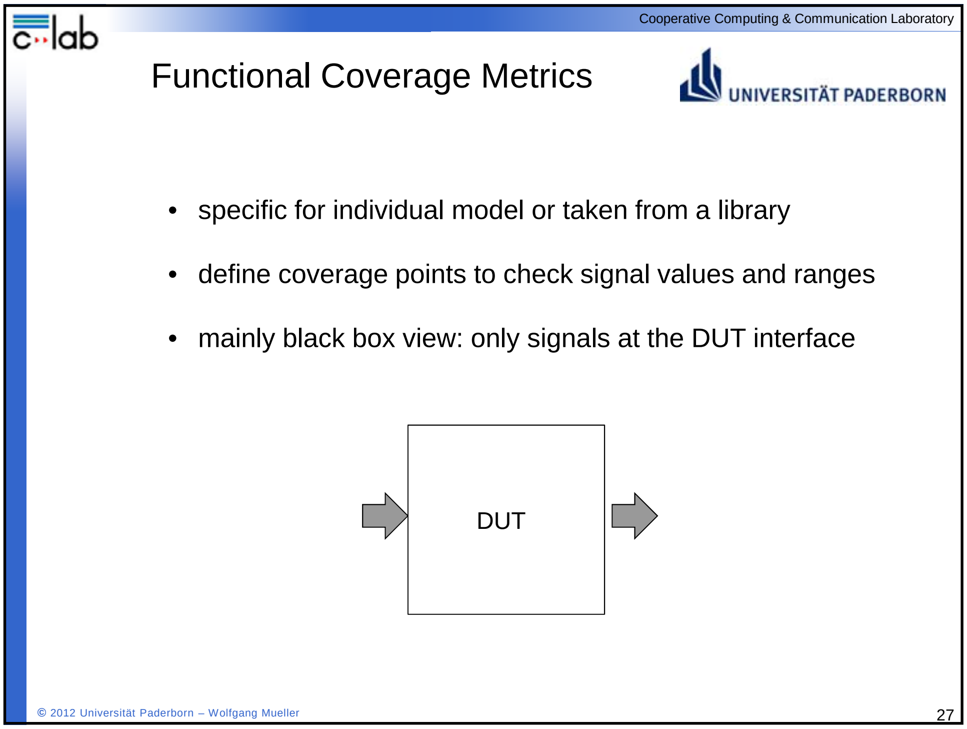
System Component Coverage

S0: execute all components at least once with actual parameters

S1: execute all components with all possible meaningful parameters

S2: execute all possible component calls at main program level

© 2012 Universität Paderborn – Wolfgang Mueller 26



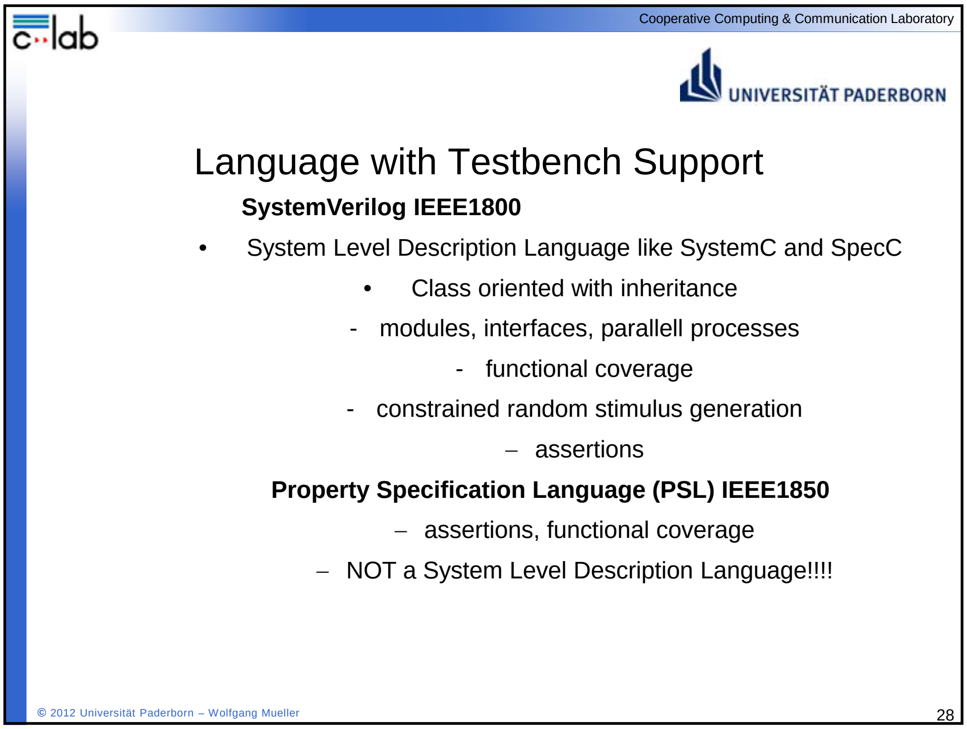
Cooperative Computing & Communication Laboratory

Functional Coverage Metrics

- specific for individual model or taken from a library
- define coverage points to check signal values and ranges
- mainly black box view: only signals at the DUT interface

© 2012 Universität Paderborn – Wolfgang Mueller

27



Cooperative Computing & Communication Laboratory

Language with Testbench Support

SystemVerilog IEEE1800

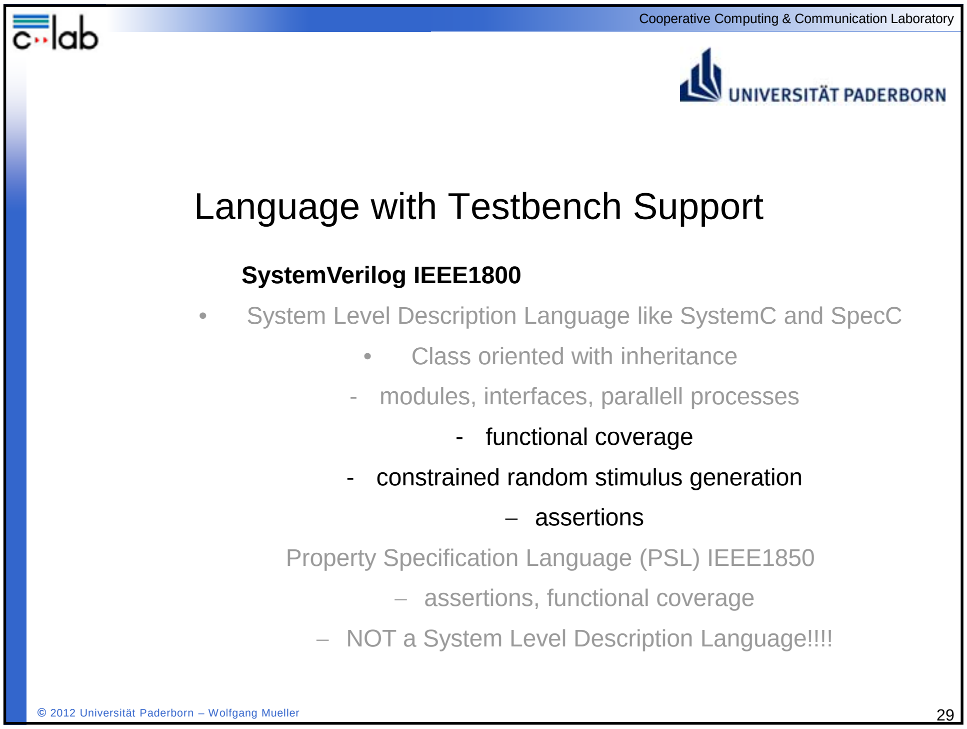
- System Level Description Language like SystemC and SpecC
  - Class oriented with inheritance
  - modules, interfaces, parallell processes
  - functional coverage
  - constrained random stimulus generation
  - assertions

Property Specification Language (PSL) IEEE1850

- assertions, functional coverage
- NOT a System Level Description Language!!!!

© 2012 Universität Paderborn – Wolfgang Mueller

28



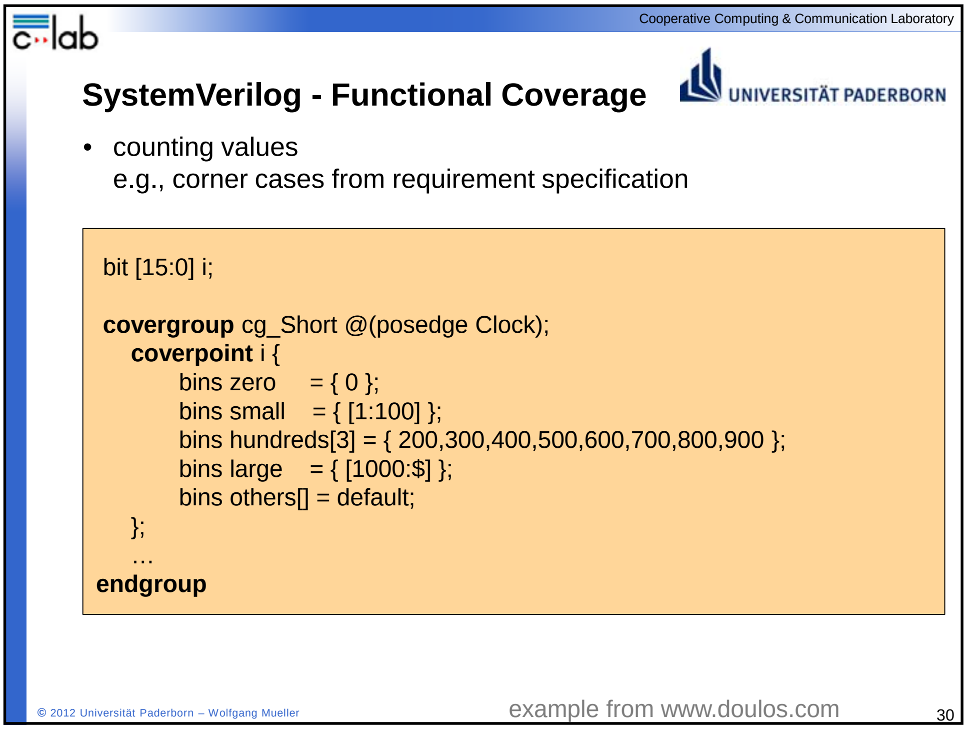
Language with Testbench Support

SystemVerilog IEEE1800

- System Level Description Language like SystemC and SpecC
  - Class oriented with inheritance
    - modules, interfaces, parallell processes
    - functional coverage
    - constrained random stimulus generation
    - assertions

Property Specification Language (PSL) IEEE1850

- assertions, functional coverage
- NOT a System Level Description Language!!!!

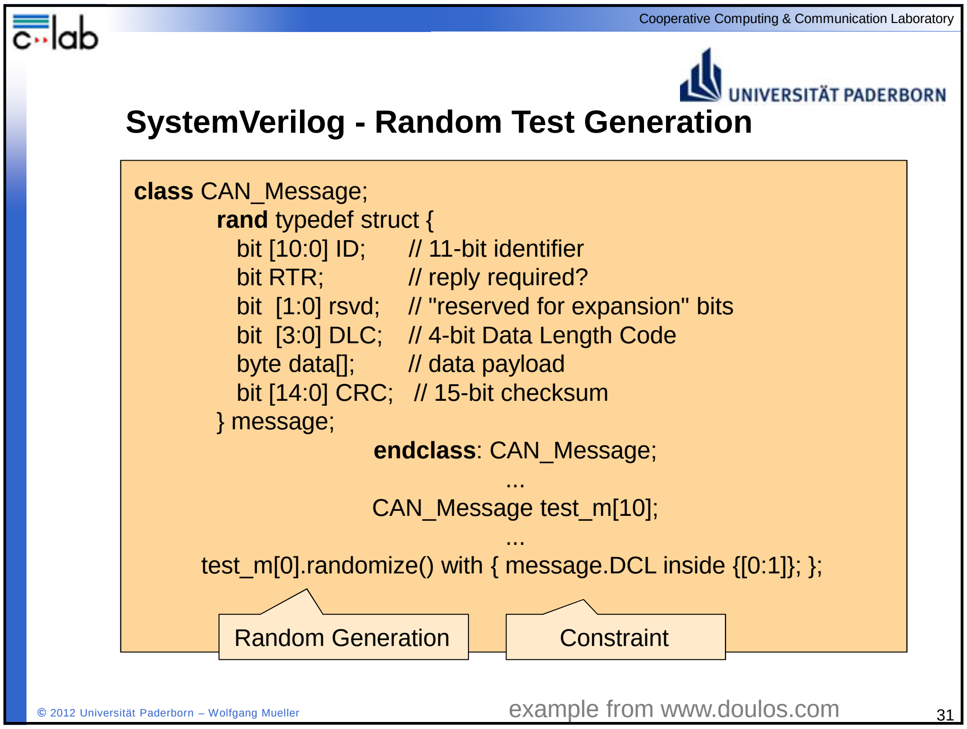


SystemVerilog - Functional Coverage

- counting values
  e.g., corner cases from requirement specification

```
bit [15:0] i;

covergroup cg_Short @(posedge Clock);
   coverpoint i {
         bins zero      = { 0 };
         bins small     = { [1:100] };
         bins hundreds[3] = { 200,300,400,500,600,700,800,900 };
         bins large     = { [1000:$] };
         bins others[] = default;
   };
   …
endgroup
```

example from www.doulos.com



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

SystemVerilog - Random Test Generation

```
class CAN_Message;
        rand typedef struct {
          bit [10:0] ID;      // 11-bit identifier
          bit RTR;            // reply required?
          bit  [1:0] rsvd;    // "reserved for expansion" bits
          bit  [3:0] DLC;     // 4-bit Data Length Code
          byte data[];        // data payload
          bit [14:0] CRC;   // 15-bit checksum
        } message;
                        endclass: CAN_Message;
                                   ...
                        CAN_Message test_m[10];
                                   ...
      test_m[0].randomize() with { message.DCL inside {[0:1]}; };
```

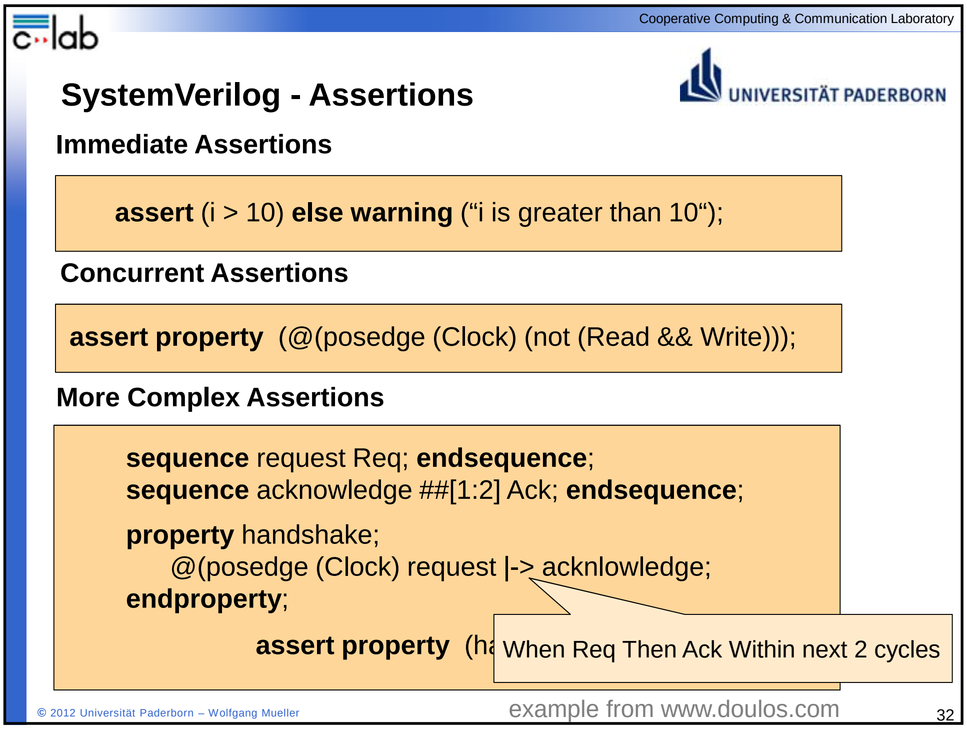
Random Generation

Constraint

© 2012 Universität Paderborn – Wolfgang Mueller

example from www.doulos.com

31



Cooperative Computing & Communication Laboratory

UNIVERSITÄT PADERBORN

SystemVerilog - Assertions

Immediate Assertions

```
assert (i > 10) else warning (“i is greater than 10“);
```

Concurrent Assertions

```
assert property  (@(posedge (Clock) (not (Read && Write)));
```

More Complex Assertions

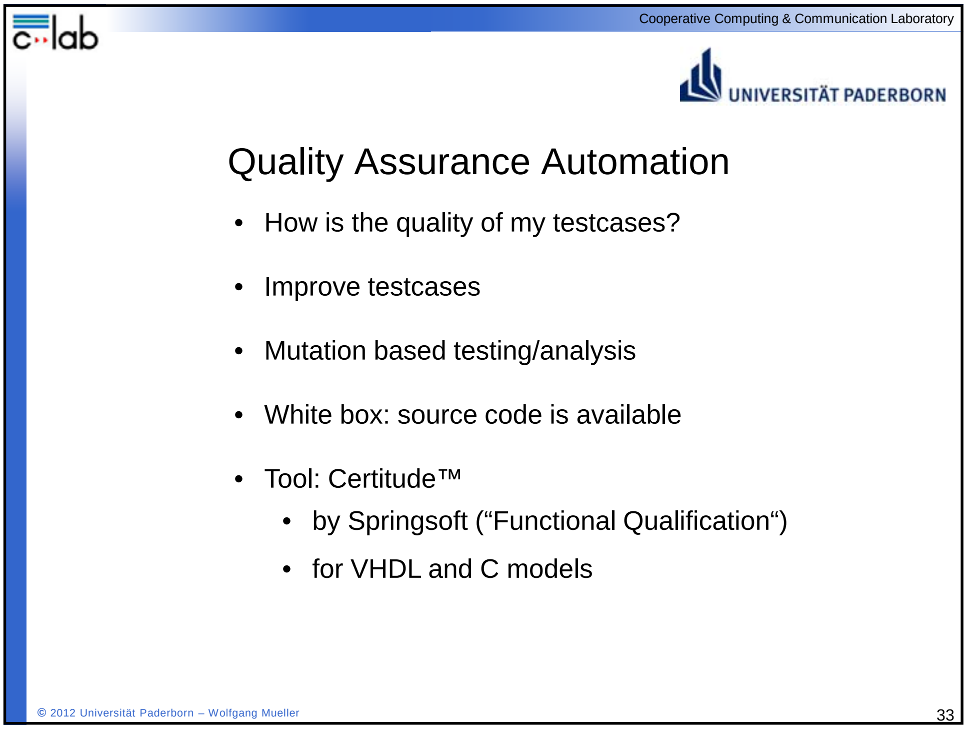
```
sequence request Req; endsequence;
sequence acknowledge ##[1:2] Ack; endsequence;

property handshake;
     @(posedge (Clock) request |-> acknlowledge;
endproperty;

                 assert property  (ha
```

When Req Then Ack Within next 2 cycles

© 2012 Universität Paderborn – Wolfgang Mueller

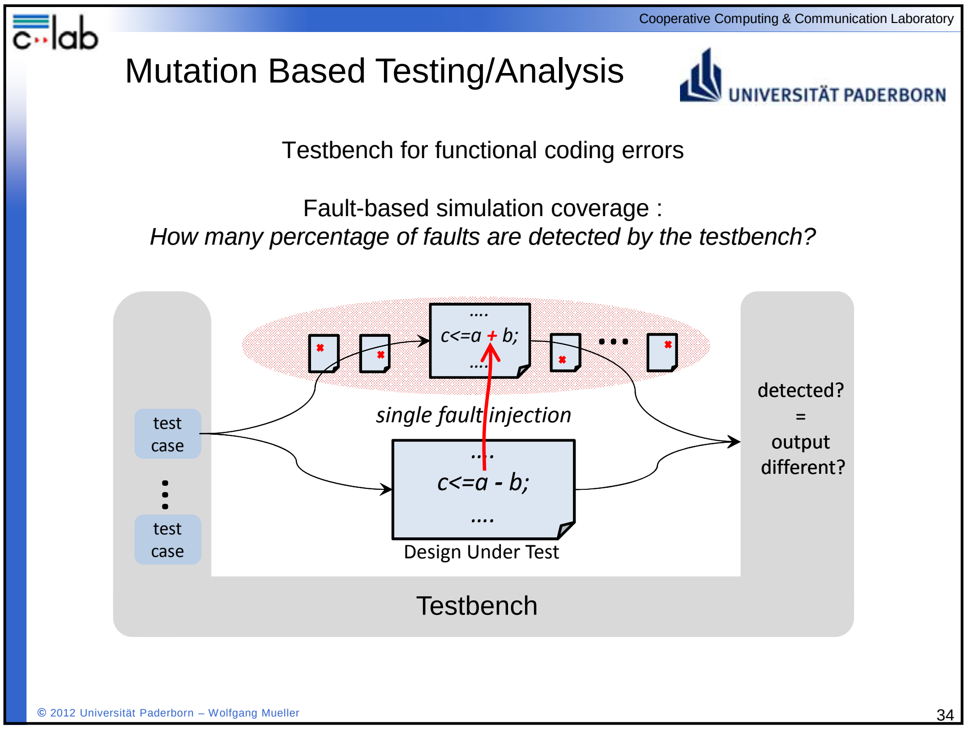
example from www.doulos.com

32



Quality Assurance Automation

- How is the quality of my testcases?
- Improve testcases
- Mutation based testing/analysis
- White box: source code is available
- Tool: Certitude™
  - by Springsoft ("Functional Qualification")
  - for VHDL and C models



Mutation Based Testing/Analysis

Testbench for functional coding errors

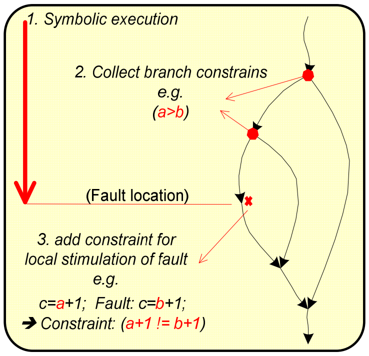
Fault-based simulation coverage :
How many percentage of faults are detected by the testbench?

Mutation Based Testing/Analysis

UNIVERSITÄT PADERBORN

Different Phases for Analysis:

- **Activate**
  Is the path to the statement/expression reachable?
- **Mutate**
  Change code
- **Propagate**
  Fault observable at the output?
- **Detect**
  Fault detected by the testbench?



Mutation Analysis with Sp

UNIVERSITÄT PADERBORN

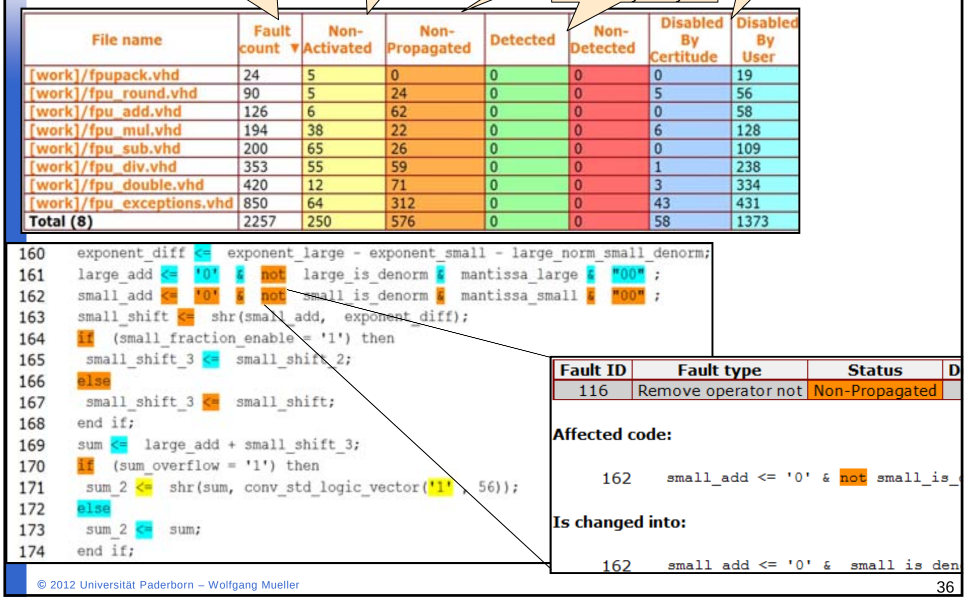
# of mutations

mutations not reachable

mutations not observable

not analyzed yet

equivalent (redundant) testcases



File name	Fault count ▼	Non-Activated	Non-Propagated	Detected	Non-Detected	Disabled By Certitude	Disabled By User
[work]/fpupack.vhd	24	5	0	0	0	0	19
[work]/fpu_round.vhd	90	5	24	0	0	5	56
[work]/fpu_add.vhd	126	6	62	0	0	0	58
[work]/fpu_mul.vhd	194	38	22	0	0	6	128
[work]/fpu_sub.vhd	200	65	26	0	0	0	109
[work]/fpu_div.vhd	353	55	59	0	0	1	238
[work]/fpu_double.vhd	420	12	71	0	0	3	334
[work]/fpu_exceptions.vhd	850	64	312	0	0	43	431
Total (8)	2257	250	576	0	0	58	1373

```
160   exponent_diff <=  exponent_large - exponent_small - large_norm_small_denorm;
161   large_add <=  '0'  &  not  large_is_denorm &  mantissa_large &  "00" ;
162   small_add <=  '0'  &  not  small_is_denorm &  mantissa_small &  "00" ;
163   small_shift <=  shr(small_add,  exponent_diff);
164   if  (small_fraction_enable = '1') then
165    small_shift_3 <=  small_shift_2;
166   else
167    small_shift_3 <=  small_shift;
168   end if;
169   sum <=  large_add + small_shift_3;
170   if  (sum_overflow = '1') then
171    sum_2 <=  shr(sum, conv_std_logic_vector('1' , 56));
172   else
173    sum_2 <=  sum;
174   end if;
```

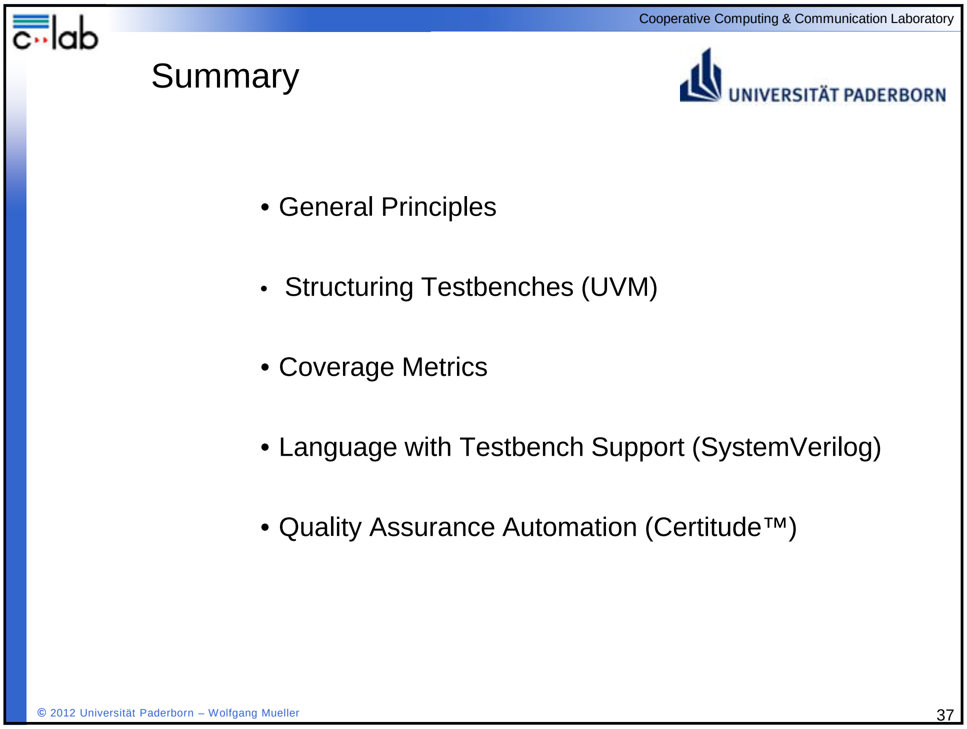
Fault ID	Fault type	Status	D
116	Remove operator not	Non-Propagated	

Affected code:

```
162    small_add <= '0' & not small_is_
```

Is changed into:

```
162    small_add <= '0' &  small_is_den
```



Cooperative Computing & Communication Laboratory

Summary

- General Principles
- Structuring Testbenches (UVM)
- Coverage Metrics
- Language with Testbench Support (SystemVerilog)
- Quality Assurance Automation (Certitude™)