Getting started with UVM or use Formal Instead?

Delivering KnowHow

Getting started with UVM or use Formal Instead?

www.doulos.com

Getting started with UVM or use Formal Instead?

Doulos Limited
Church Hatch, 22 Market Place,
Ringwood, Hampshire, BH24 1AW, UK

Tel: +44 (0) 1425 471223

Email: info@doulos.com

Doulos
6203 San Ignacio Avenue, Suite 110,
San Jose, CA 95119, USA

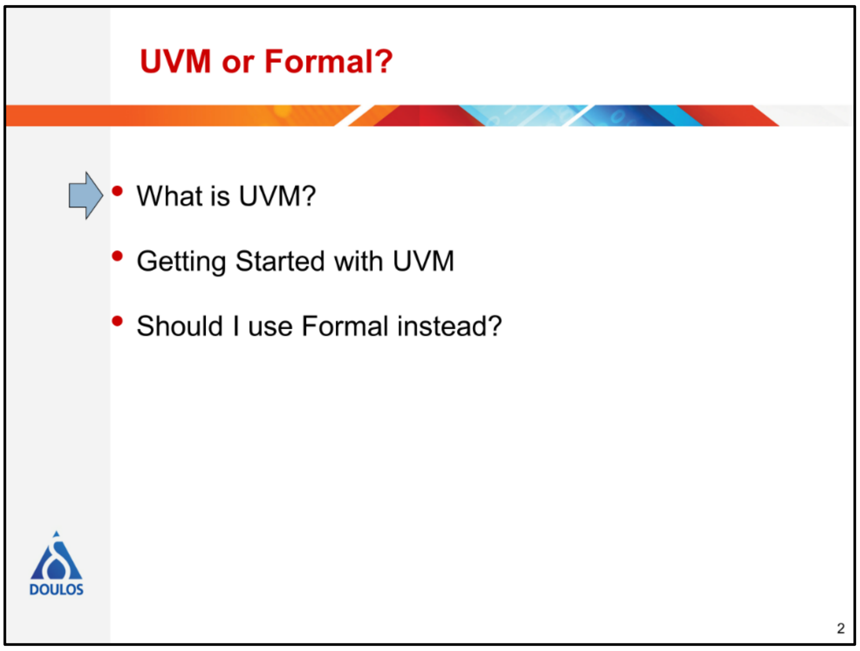
Tel: 1-888-GO DOULOS

Email: info.usa@doulos.com

www.doulos.com

Contents

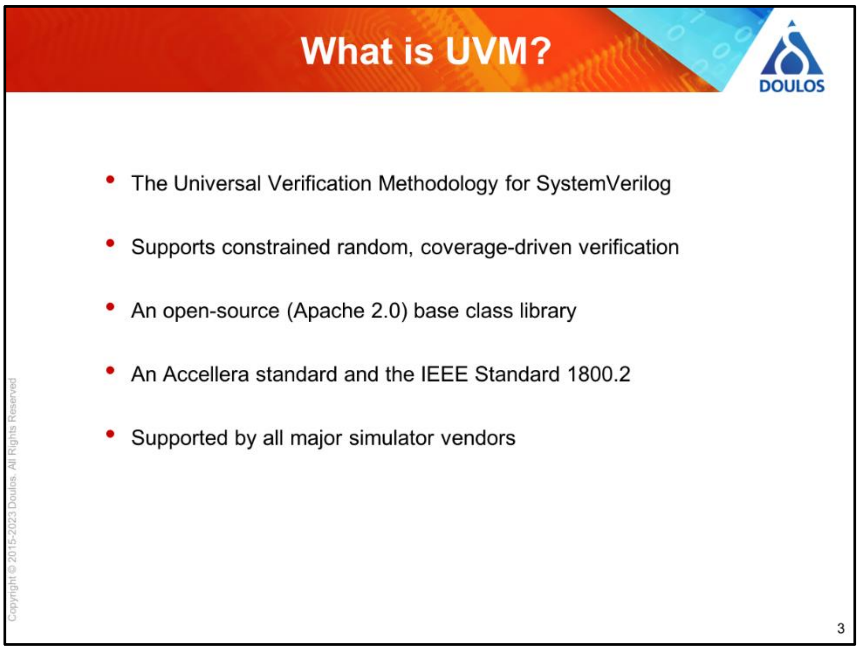
What is UVM?



UVM or Formal?

- What is UVM?
- Getting Started with UVM
- Should I use Formal instead?

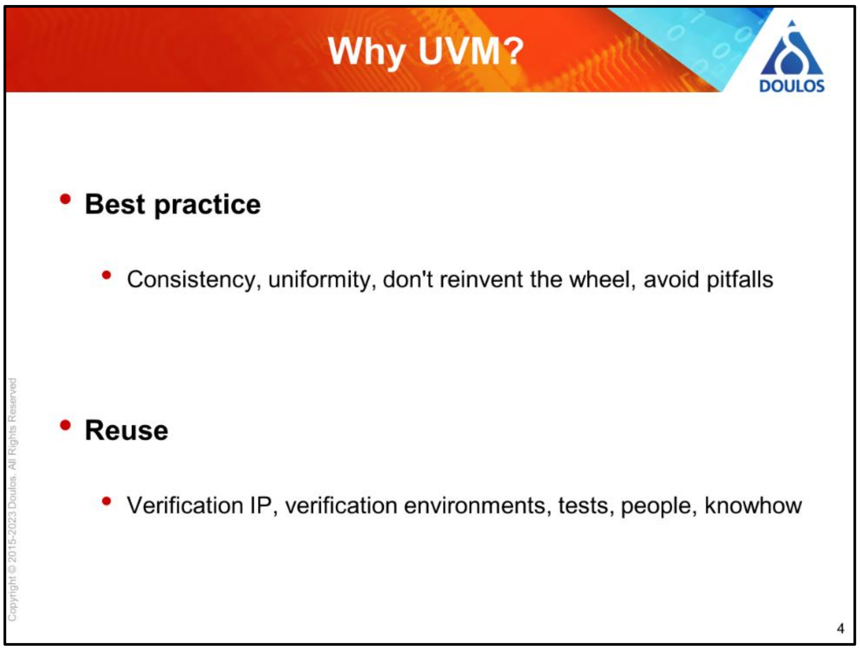
2



What is UVM?

- The Universal Verification Methodology for SystemVerilog
- Supports constrained random, coverage-driven verification
- An open-source (Apache 2.0) base class library
- An Accellera standard and the IEEE Standard 1800.2
- Supported by all major simulator vendors

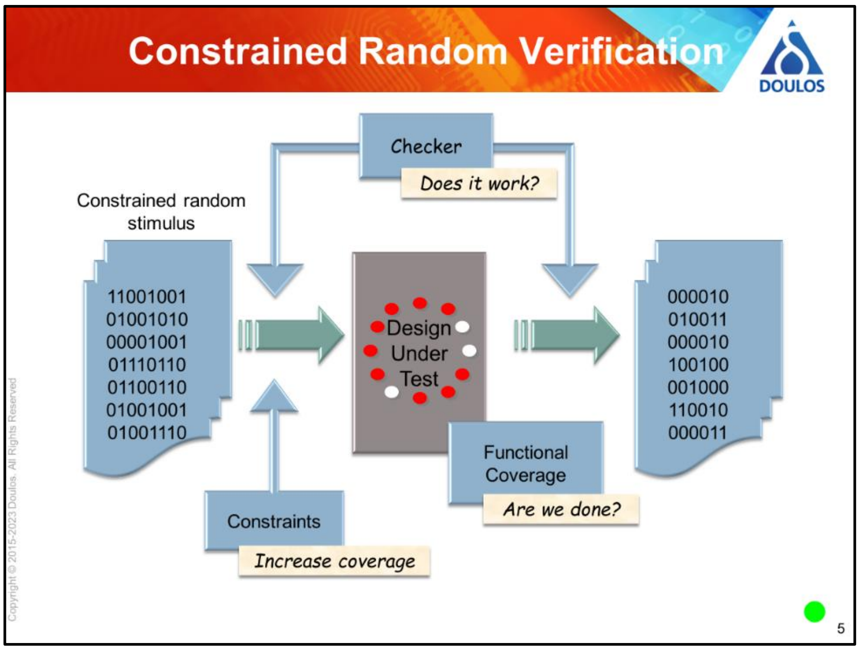
3



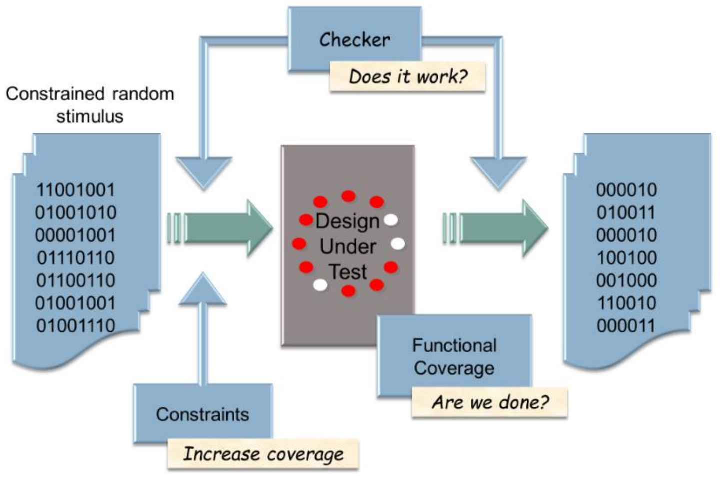
Why UVM?

- **Best practice**
  - Consistency, uniformity, don't reinvent the wheel, avoid pitfalls
- **Reuse**
  - Verification IP, verification environments, tests, people, knowhow

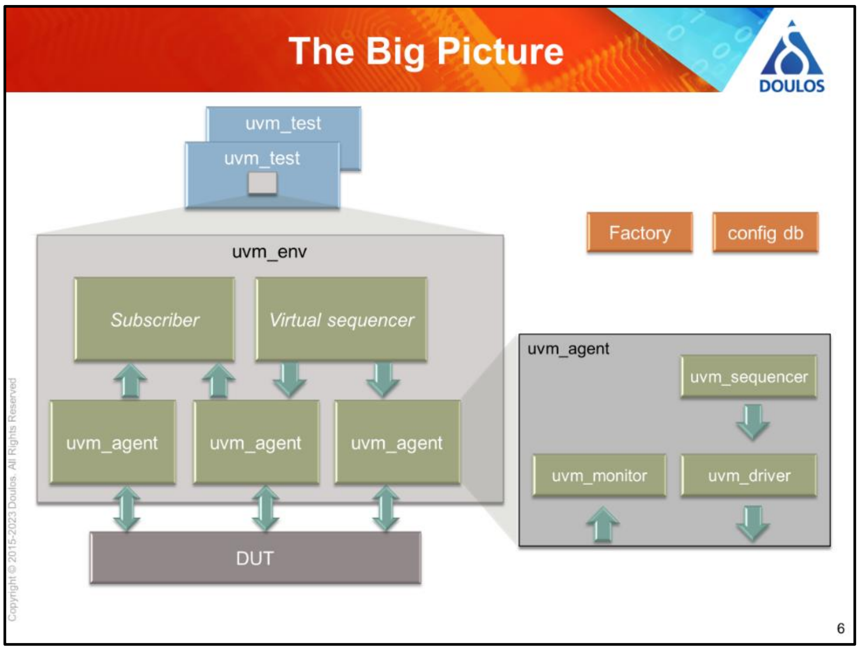
4



Constrained Random Verification

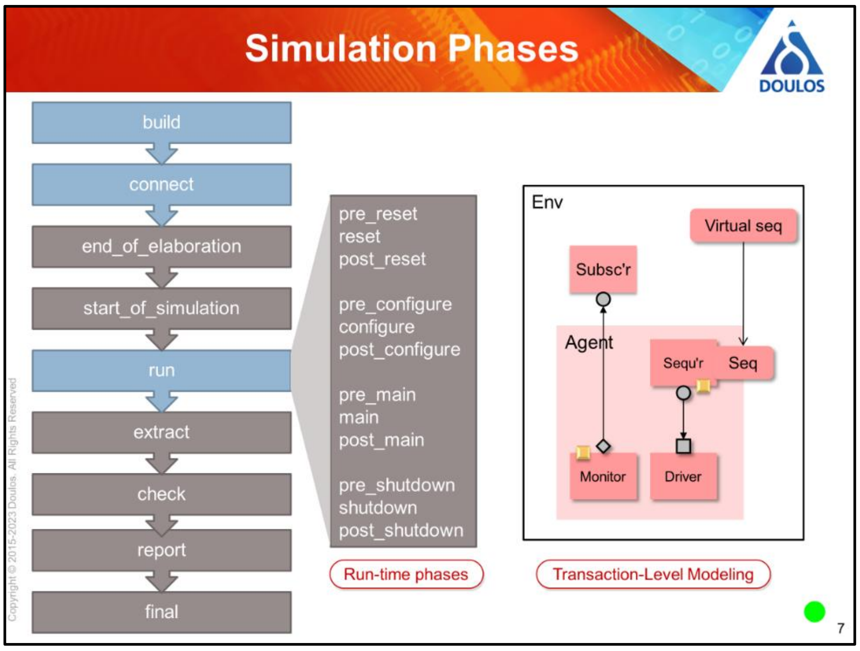


5



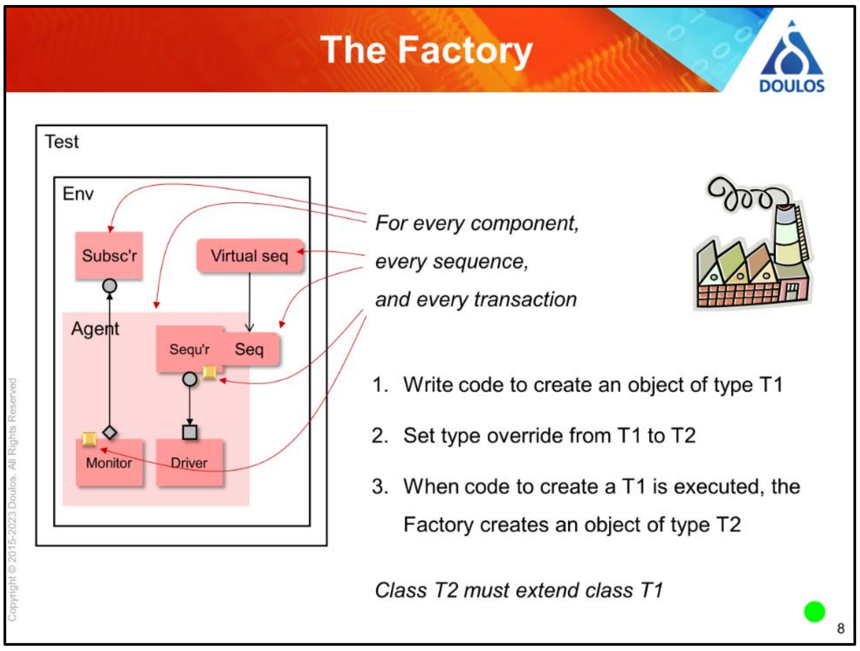
The Big Picture

6

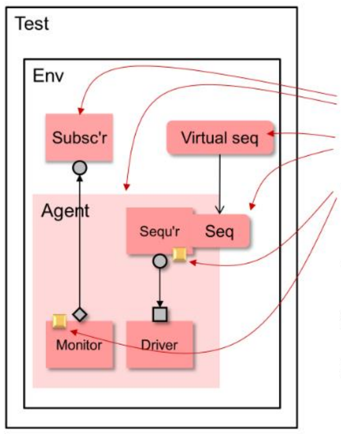


Simulation Phases

7



The Factory



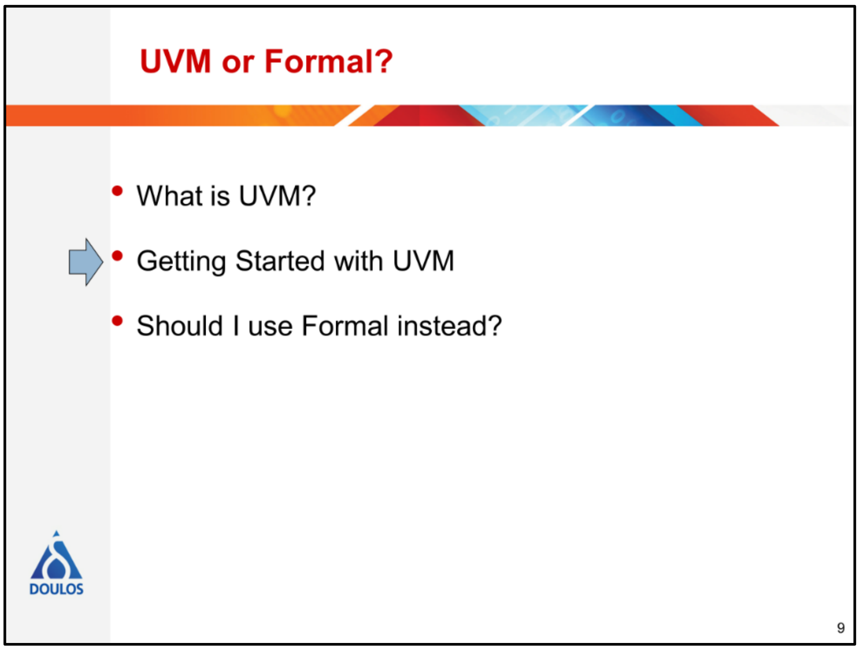
For every component,
every sequence,
and every transaction

1. Write code to create an object of type T1
2. Set type override from T1 to T2
3. When code to create a T1 is executed, the Factory creates an object of type T2

Class T2 must extend class T1

8

Getting Started with UVM

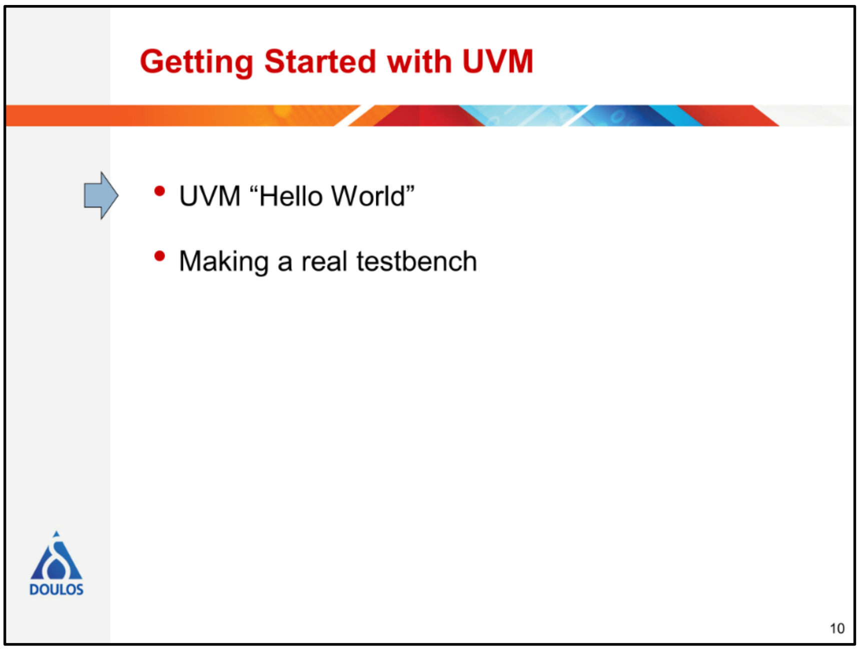


UVM or Formal?

- What is UVM?
- Getting Started with UVM
- Should I use Formal instead?

9

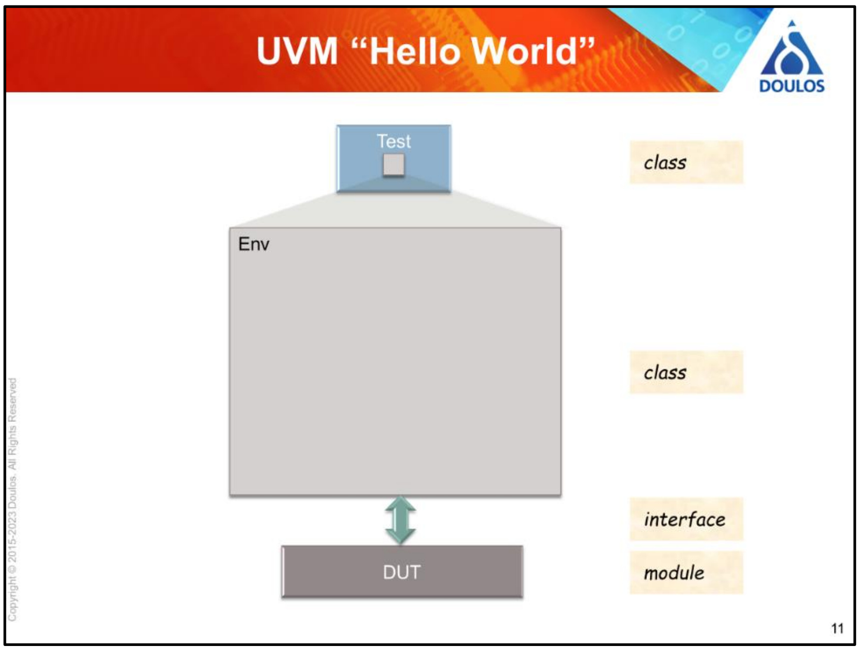
UVM “Hello World”

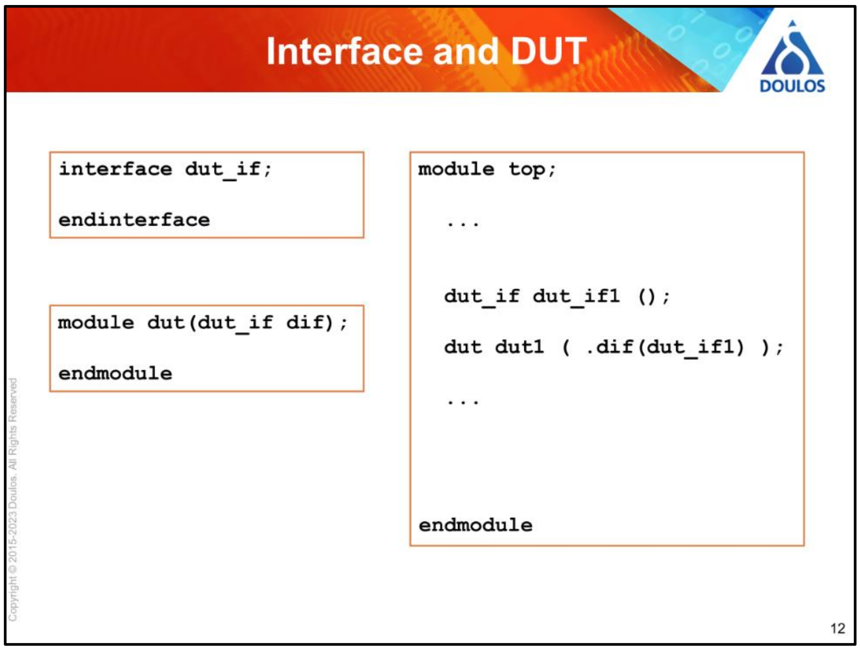


Getting Started with UVM

- UVM “Hello World”
- Making a real testbench

10





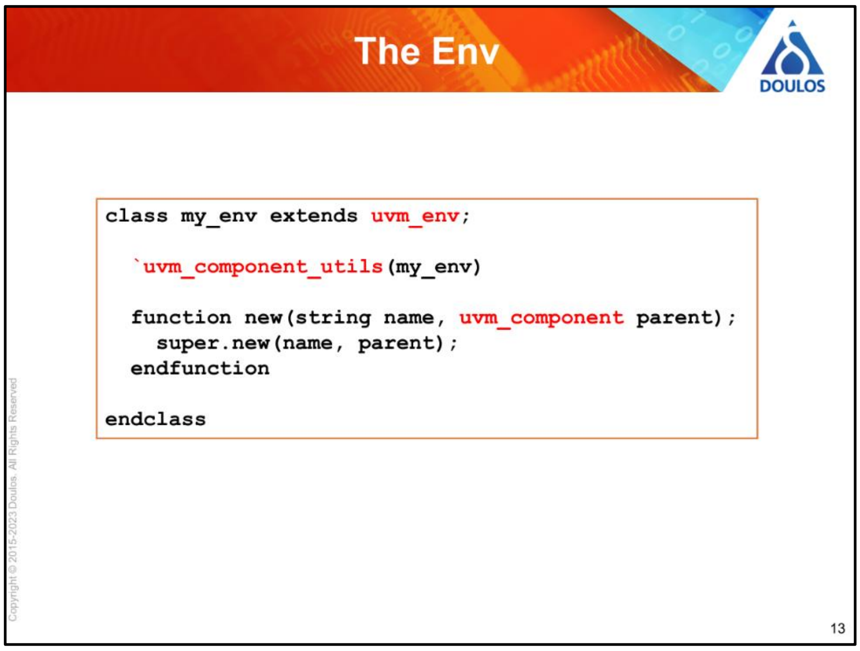
Interface and DUT

```
interface dut_if;

endinterface
```

```
module dut(dut_if dif);

endmodule
```

```
module top;

  ...


  dut_if dut_if1 ();

  dut dut1 ( .dif(dut_if1) );

  ...



endmodule
```

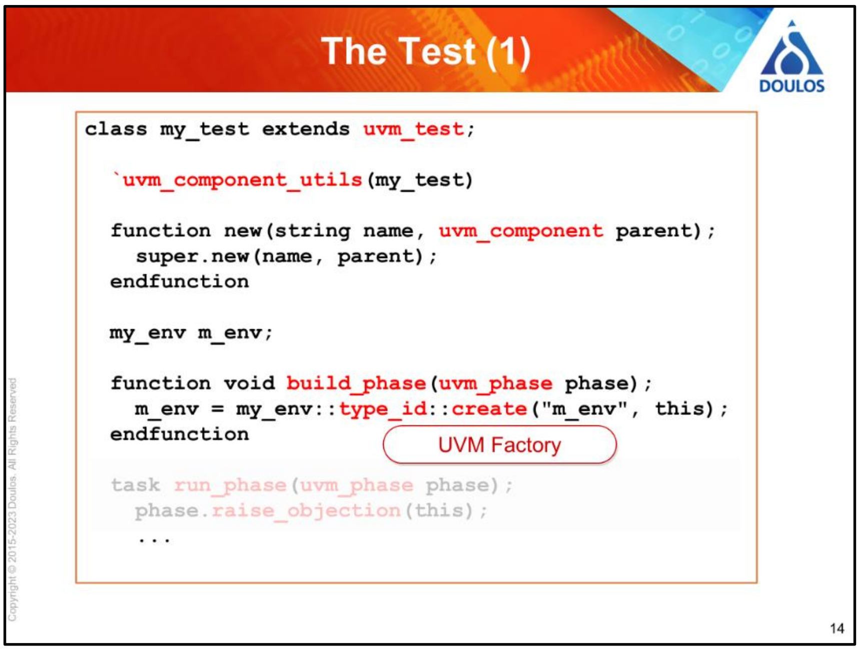
12



The Env

```
class my_env extends uvm_env;

  `uvm_component_utils(my_env)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

endclass
```

13

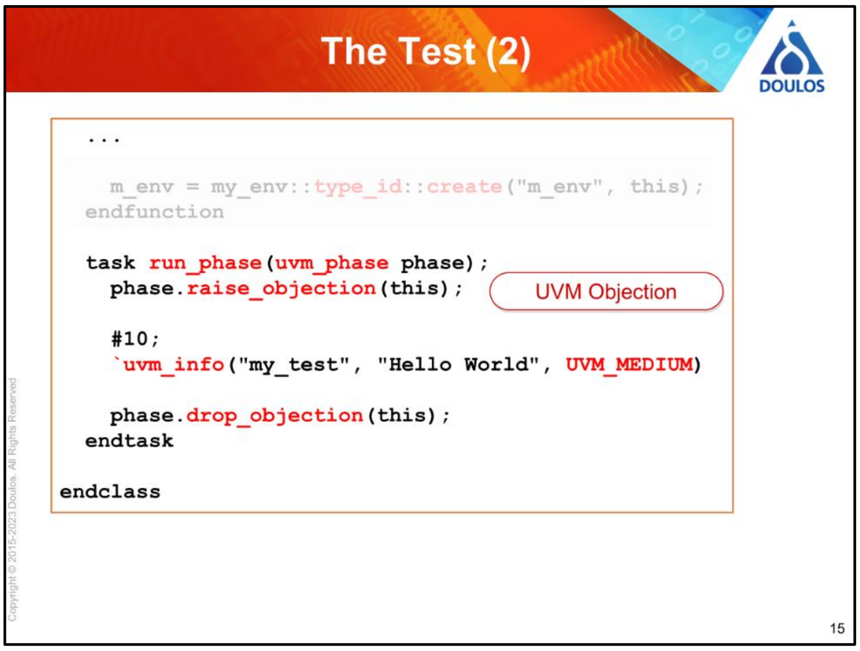


The Test (1)

```
class my_test extends uvm_test;

  `uvm_component_utils(my_test)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  my_env m_env;

  function void build_phase(uvm_phase phase);
    m_env = my_env::type_id::create("m_env", this);
  endfunction

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    ...
```

UVM Factory

14

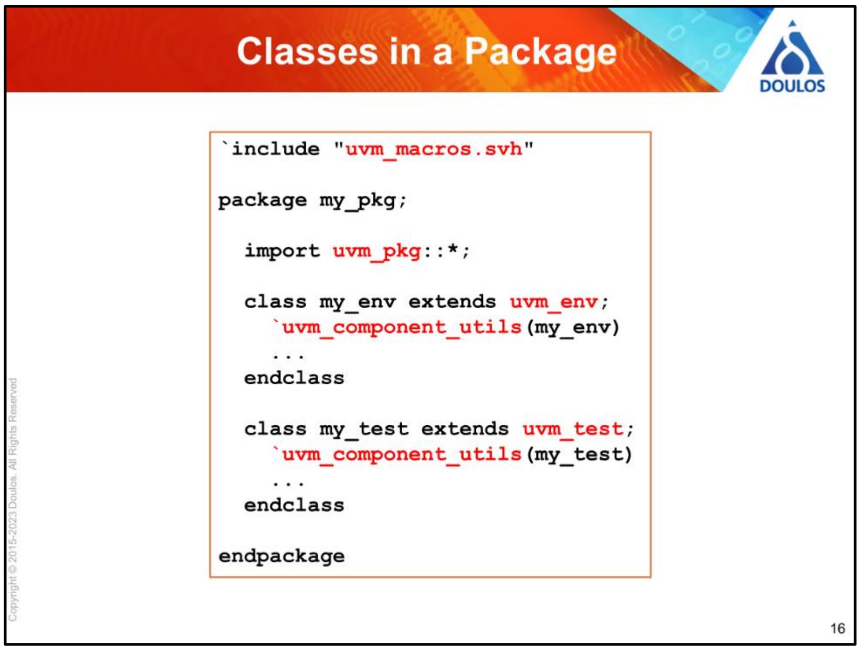


The Test (2)

```
  ...

    m_env = my_env::type_id::create("m_env", this);
  endfunction

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);

    #10;
    `uvm_info("my_test", "Hello World", UVM_MEDIUM)

    phase.drop_objection(this);
  endtask

endclass
```

UVM Objection

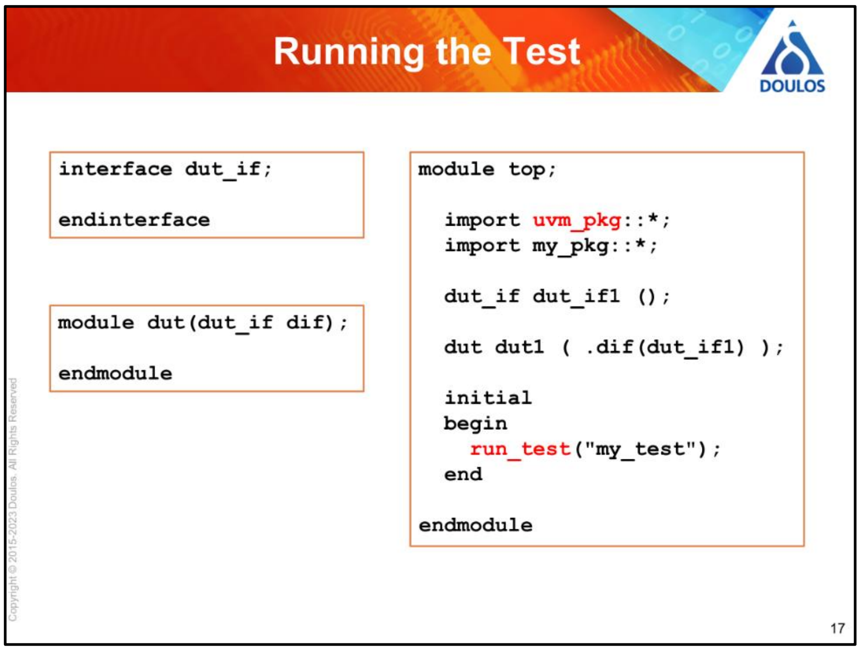
15



Classes in a Package

```
`include "uvm_macros.svh"

package my_pkg;

  import uvm_pkg::*;

  class my_env extends uvm_env;
    `uvm_component_utils(my_env)
    ...
  endclass

  class my_test extends uvm_test;
    `uvm_component_utils(my_test)
    ...
  endclass

endpackage
```

16



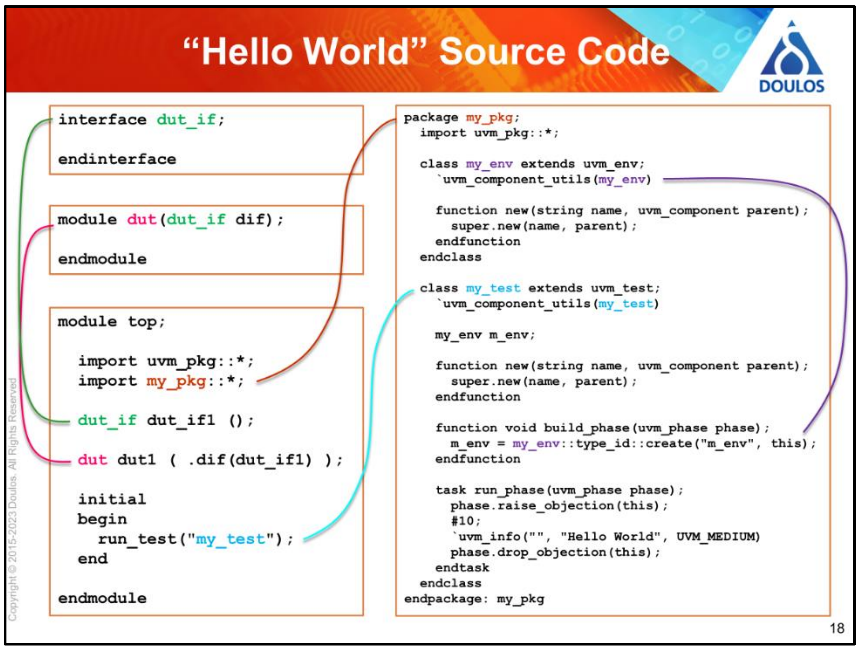
Running the Test

```
interface dut_if;

endinterface
```

```
module dut(dut_if dif);

endmodule
```

```
module top;

  import uvm_pkg::*;
  import my_pkg::*;

  dut_if dut_if1 ();

  dut dut1 ( .dif(dut_if1) );

  initial
  begin
    run_test("my_test");
  end

endmodule
```

17



“Hello World” Source Code

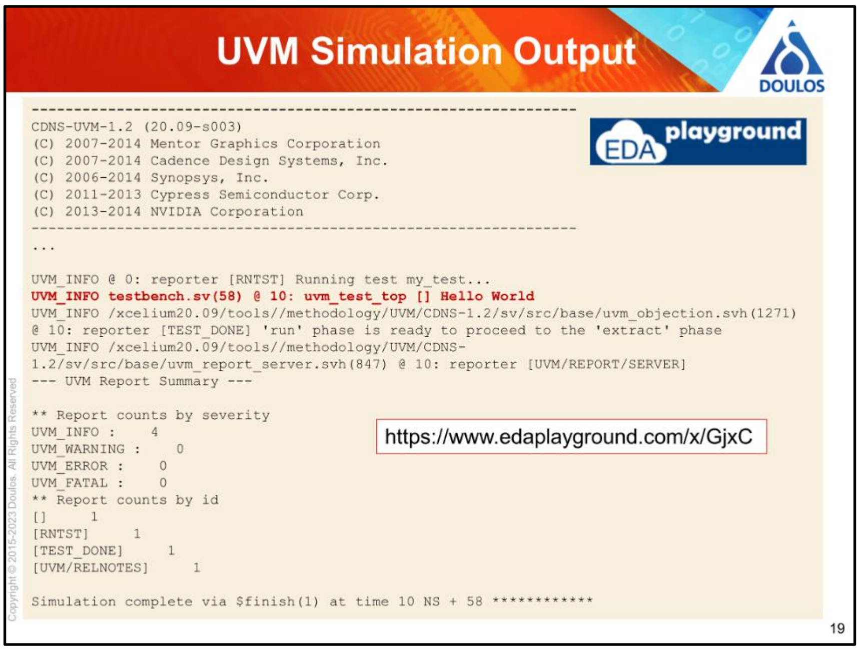
```
interface dut_if;

endinterface
```

```
module dut(dut_if dif);

endmodule
```

```
module top;

  import uvm_pkg::*;
  import my_pkg::*;

  dut_if dut_if1 ();

  dut dut1 ( .dif(dut_if1) );

  initial
  begin
    run_test("my_test");
  end

endmodule
```

```
package my_pkg;
  import uvm_pkg::*;

  class my_env extends uvm_env;
    `uvm_component_utils(my_env)

    function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction
  endclass

  class my_test extends uvm_test;
    `uvm_component_utils(my_test)

    my_env m_env;

    function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      m_env = my_env::type_id::create("m_env", this);
    endfunction

    task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      #10;
      `uvm_info("", "Hello World", UVM_MEDIUM)
      phase.drop_objection(this);
    endtask
  endclass
endpackage: my_pkg
```

18



UVM Simulation Output

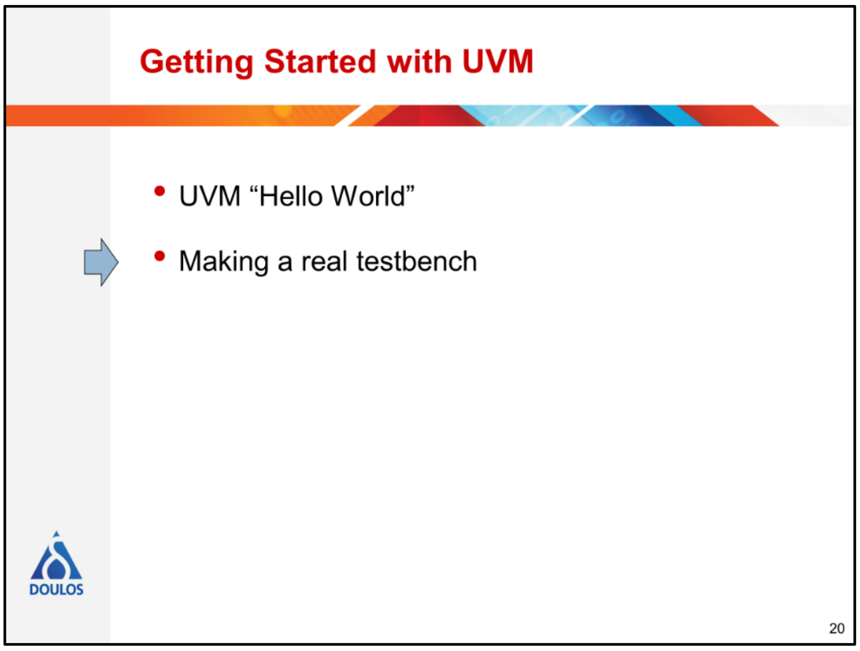
```
----------------------------------------------------------------
CDNS-UVM-1.2 (20.09-s003)
(C) 2007-2014 Mentor Graphics Corporation
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------
...

UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO testbench.sv(58) @ 10: uvm_test_top [] Hello World
UVM_INFO /xcelium20.09/tools//methodology/UVM/CDNS-1.2/sv/src/base/uvm_objection.svh(1271)
@ 10: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO /xcelium20.09/tools//methodology/UVM/CDNS-
1.2/sv/src/base/uvm_report_server.svh(847) @ 10: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :    4
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[]     1
[RNTST]     1
[TEST_DONE]     1
[UVM/RELNOTES]     1

Simulation complete via $finish(1) at time 10 NS + 58 ************
```

EDA playground

https://www.edaplayground.com/x/GjxC

19

Making a real testbench

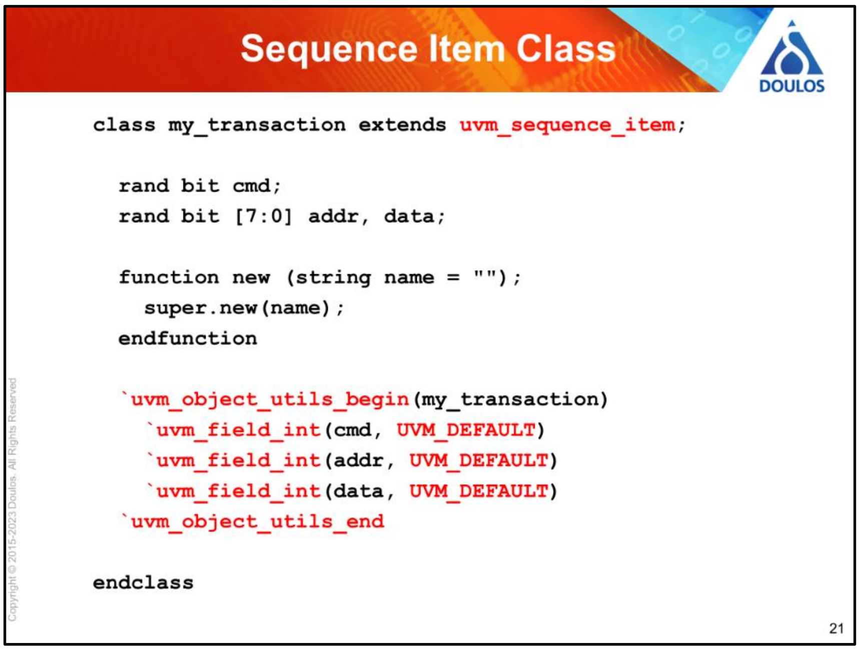


Getting Started with UVM

- UVM “Hello World”
- Making a real testbench

DOULOS

20

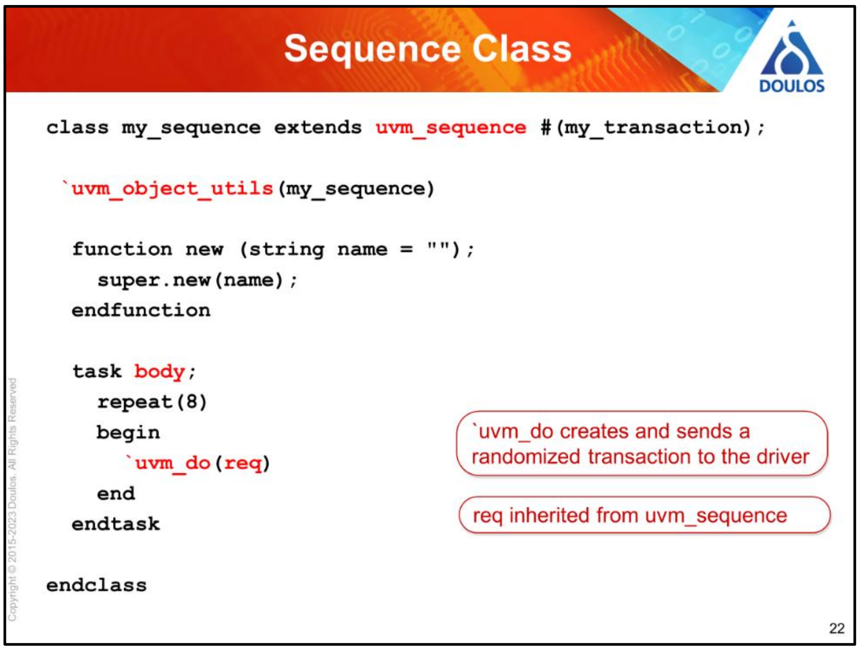


Sequence Item Class

DOULOS

```
class my_transaction extends uvm_sequence_item;

  rand bit cmd;
  rand bit [7:0] addr, data;

  function new (string name = "");
    super.new(name);
  endfunction

  `uvm_object_utils_begin(my_transaction)
    `uvm_field_int(cmd, UVM_DEFAULT)
    `uvm_field_int(addr, UVM_DEFAULT)
    `uvm_field_int(data, UVM_DEFAULT)
  `uvm_object_utils_end

endclass
```

21

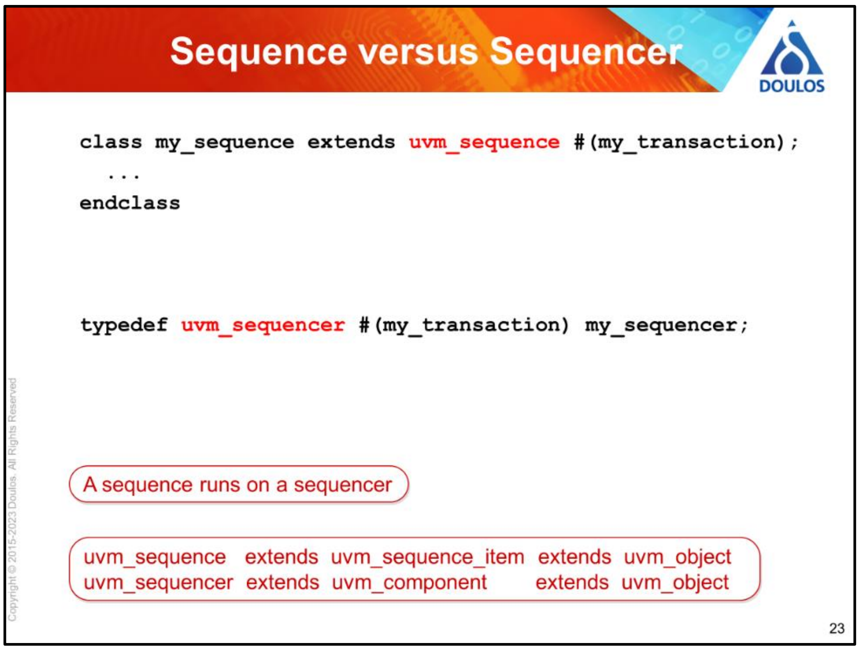


Sequence Class

```
class my_sequence extends uvm_sequence #(my_transaction);

 `uvm_object_utils(my_sequence)

  function new (string name = "");
    super.new(name);
  endfunction

  task body;
    repeat(8)
    begin
      `uvm_do(req)
    end
  endtask

endclass
```

`uvm_do creates and sends a randomized transaction to the driver

req inherited from uvm_sequence

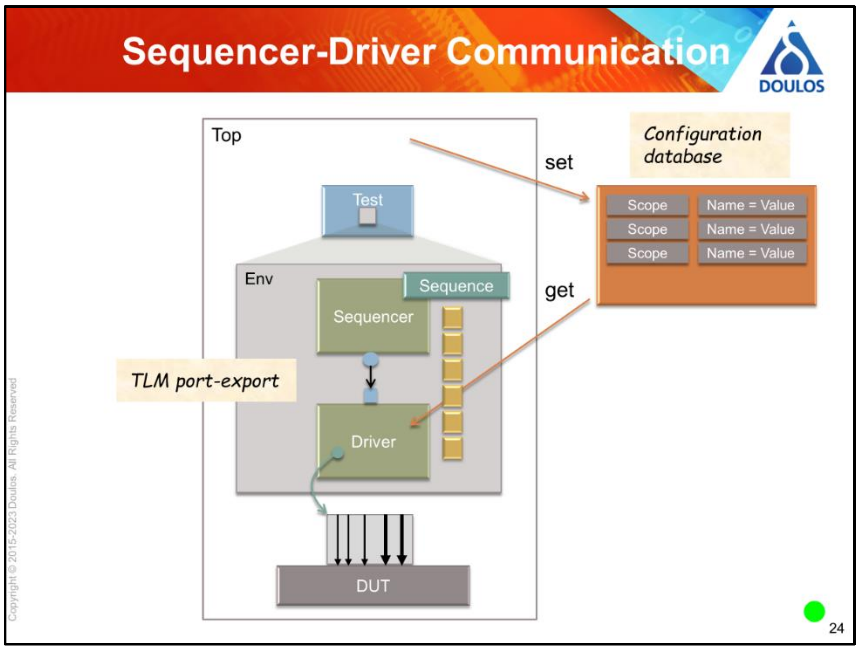


Sequence versus Sequencer

```
class my_sequence extends uvm_sequence #(my_transaction);
  ...
endclass



typedef uvm_sequencer #(my_transaction) my_sequencer;
```

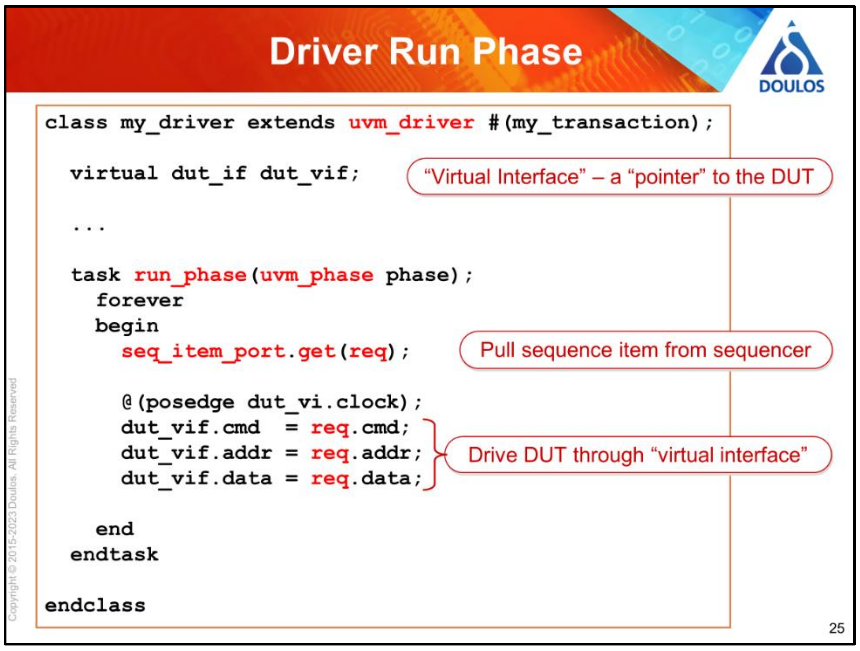
A sequence runs on a sequencer

```
uvm_sequence   extends  uvm_sequence_item  extends  uvm_object
uvm_sequencer  extends  uvm_component      extends  uvm_object
```



Sequencer-Driver Communication

24



Driver Run Phase

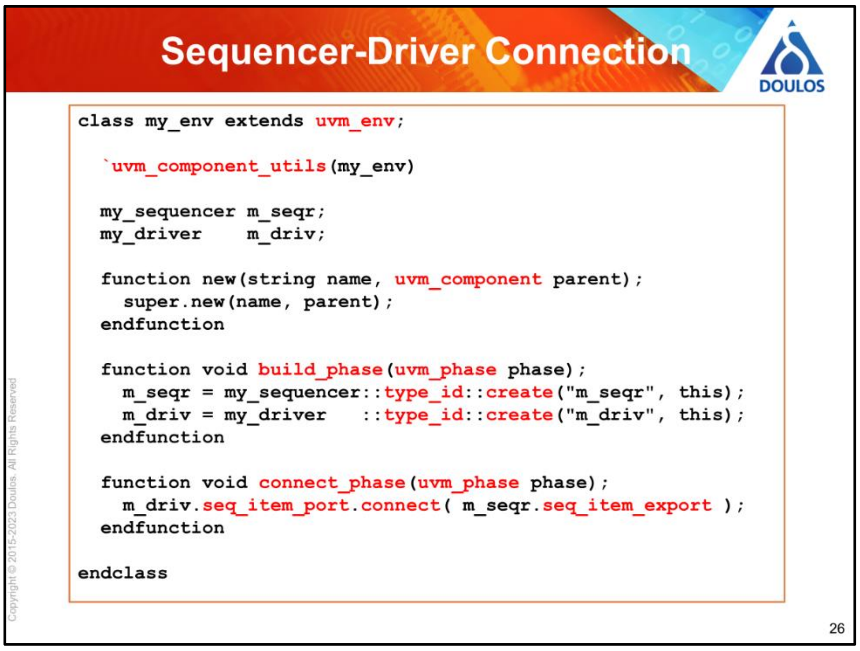
```
class my_driver extends uvm_driver #(my_transaction);

  virtual dut_if dut_vif;

  ...

  task run_phase(uvm_phase phase);
    forever
    begin
      seq_item_port.get(req);

      @(posedge dut_vi.clock);
      dut_vif.cmd  = req.cmd;
      dut_vif.addr = req.addr;
      dut_vif.data = req.data;

    end
  endtask

endclass
```

"Virtual Interface" – a "pointer" to the DUT

Pull sequence item from sequencer

Drive DUT through "virtual interface"

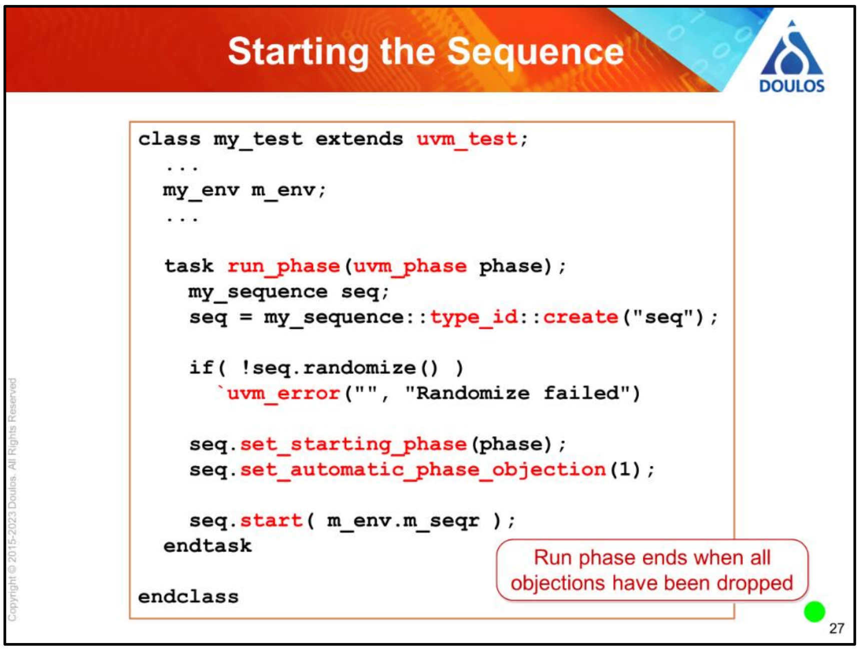
25



Sequencer-Driver Connection

```
class my_env extends uvm_env;

  `uvm_component_utils(my_env)

  my_sequencer m_seqr;
  my_driver    m_driv;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    m_seqr = my_sequencer::type_id::create("m_seqr", this);
    m_driv = my_driver   ::type_id::create("m_driv", this);
  endfunction

  function void connect_phase(uvm_phase phase);
    m_driv.seq_item_port.connect( m_seqr.seq_item_export );
  endfunction

endclass
```

26

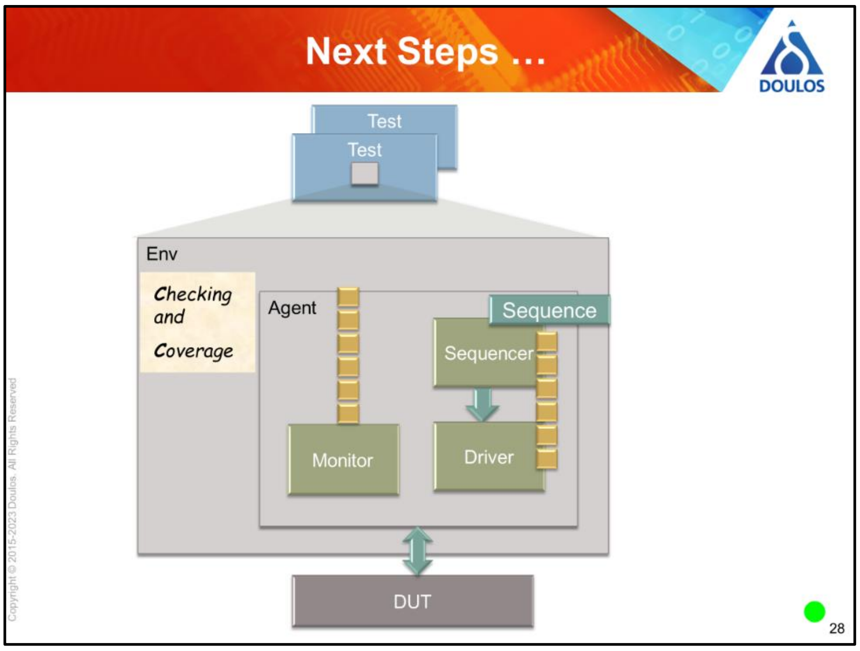


Starting the Sequence

```
class my_test extends uvm_test;
  ...
  my_env m_env;
  ...

  task run_phase(uvm_phase phase);
    my_sequence seq;
    seq = my_sequence::type_id::create("seq");

    if( !seq.randomize() )
      `uvm_error("", "Randomize failed")

    seq.set_starting_phase(phase);
    seq.set_automatic_phase_objection(1);

    seq.start( m_env.m_seqr );
  endtask

endclass
```

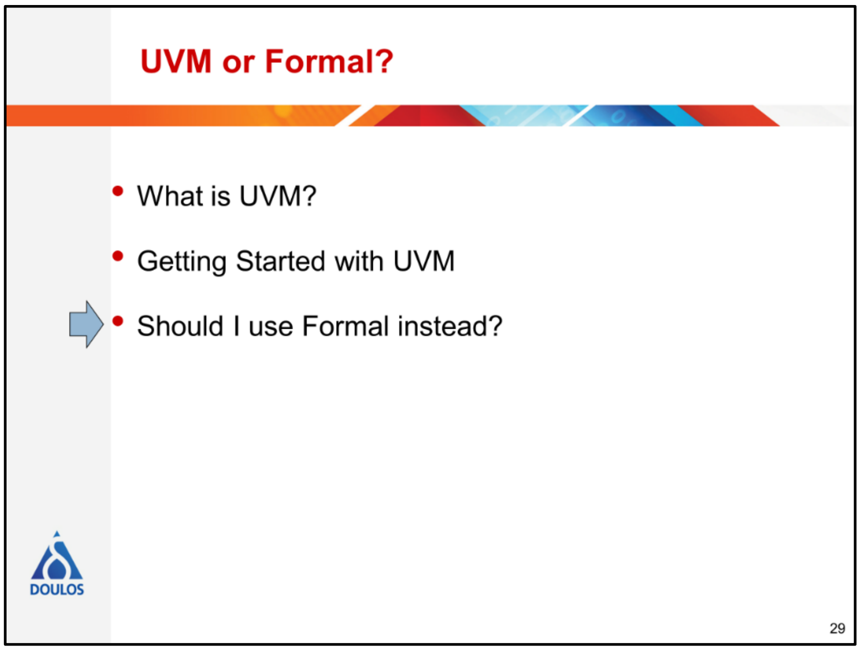
Run phase ends when all objections have been dropped

27



Next Steps …

Should I use Formal Instead?

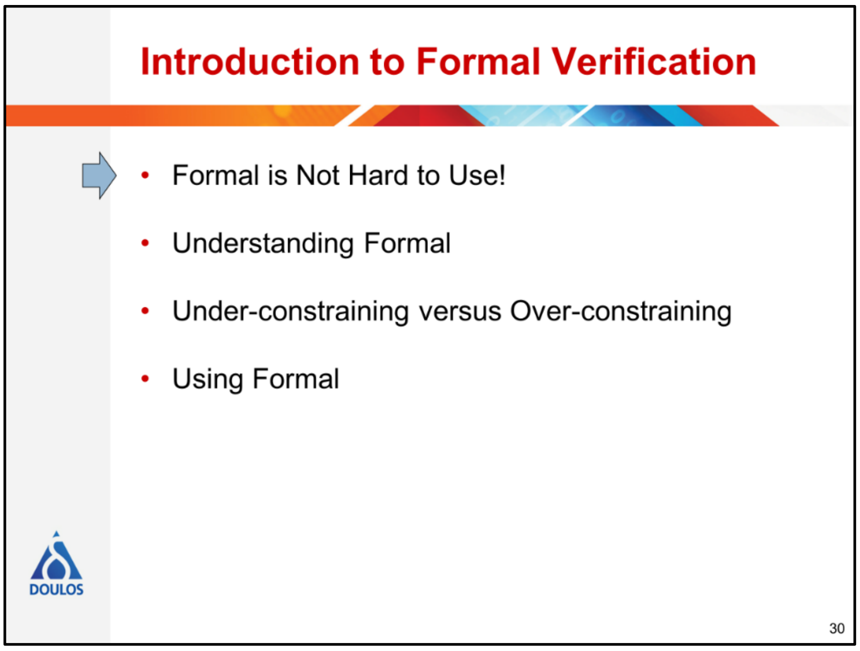


UVM or Formal?

- What is UVM?
- Getting Started with UVM
- Should I use Formal instead?

29

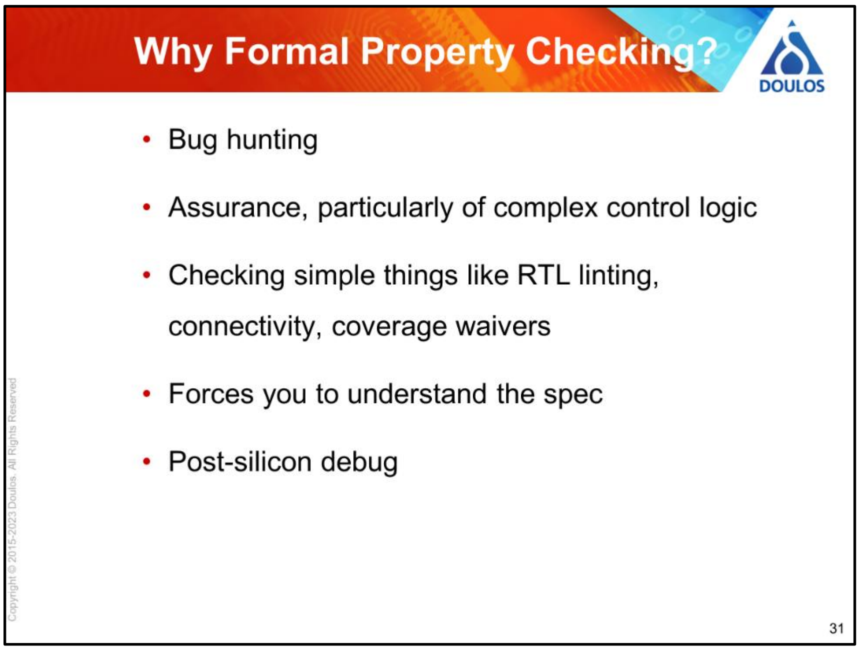
Formal is Not Hard to Use



Introduction to Formal Verification

- Formal is Not Hard to Use!
- Understanding Formal
- Under-constraining versus Over-constraining
- Using Formal

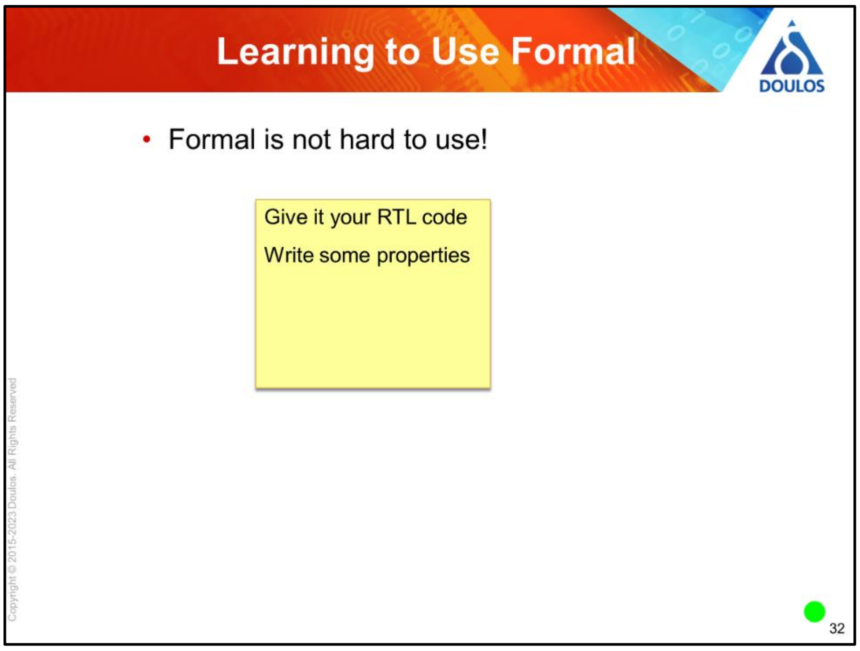
30



Why Formal Property Checking?

- Bug hunting
- Assurance, particularly of complex control logic
- Checking simple things like RTL linting, connectivity, coverage waivers
- Forces you to understand the spec
- Post-silicon debug

31



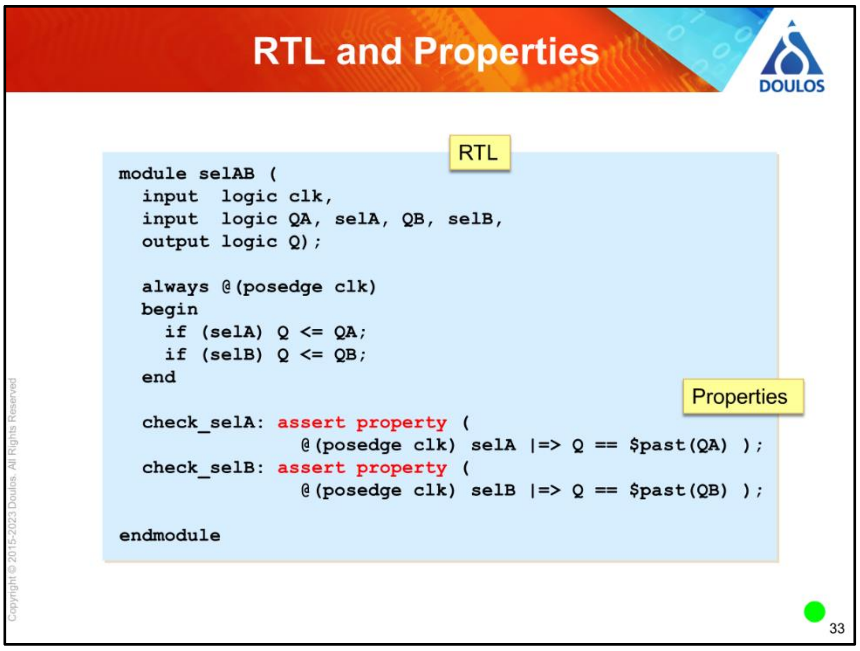
Learning to Use Formal

- Formal is not hard to use!

Give it your RTL code

Write some properties

32



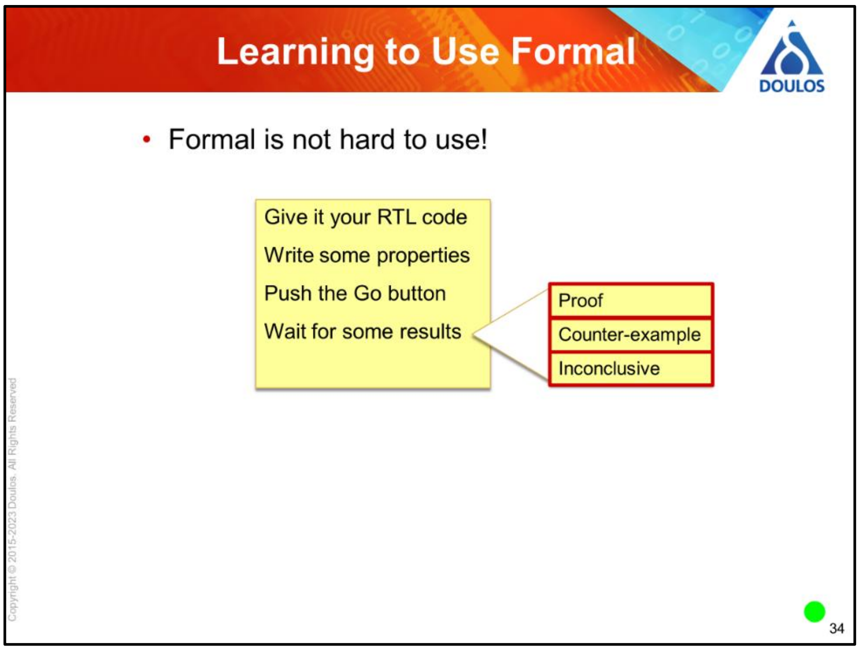
RTL and Properties

RTL

```
module selAB (
  input  logic clk,
  input  logic QA, selA, QB, selB,
  output logic Q);

  always @(posedge clk)
  begin
    if (selA) Q <= QA;
    if (selB) Q <= QB;
  end

  check_selA: assert property (
              @(posedge clk) selA |=> Q == $past(QA) );
  check_selB: assert property (
              @(posedge clk) selB |=> Q == $past(QB) );

endmodule
```

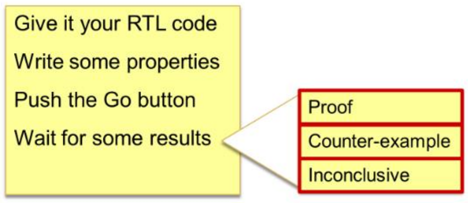
Properties

33

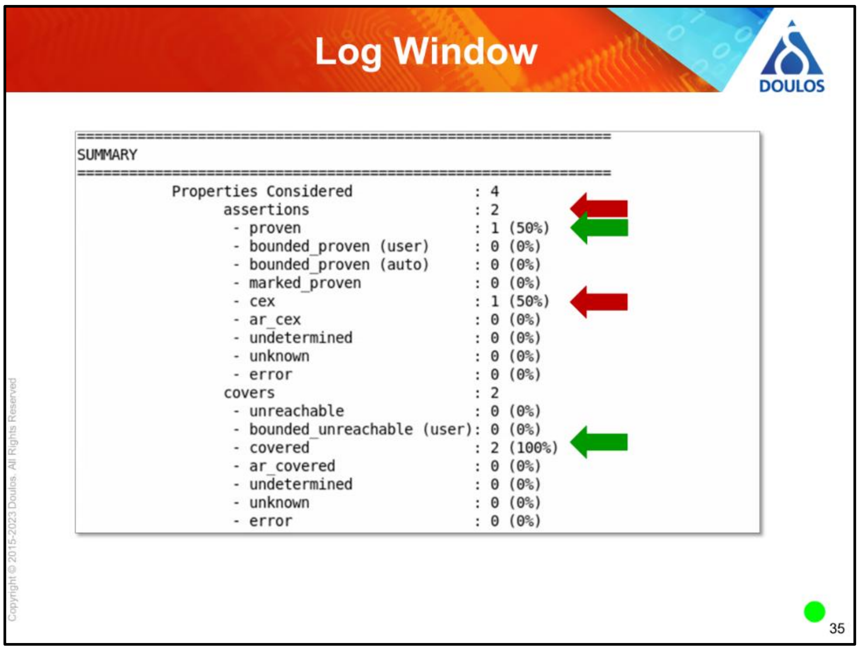


Learning to Use Formal

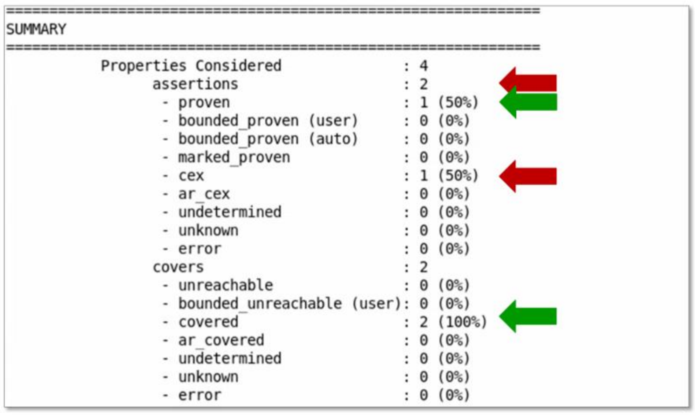
- Formal is not hard to use!



34



Log Window

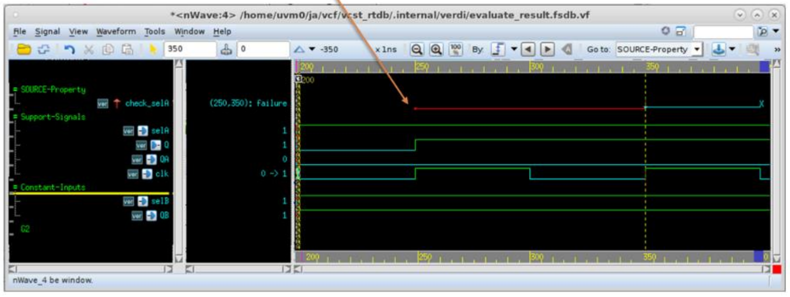


```
=================================================================
SUMMARY
=================================================================
           Properties Considered              : 4
                 assertions                   : 2
                  - proven                    : 1 (50%)
                  - bounded_proven (user)     : 0 (0%)
                  - bounded_proven (auto)     : 0 (0%)
                  - marked_proven             : 0 (0%)
                  - cex                       : 1 (50%)
                  - ar_cex                    : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
                 covers                       : 2
                  - unreachable               : 0 (0%)
                  - bounded_unreachable (user): 0 (0%)
                  - covered                   : 2 (100%)
                  - ar_covered                : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
```

35

Counter-Example (CEX)

Property fails when SelA = SelB = 1 and QA != QB



36

Learning to Use Formal

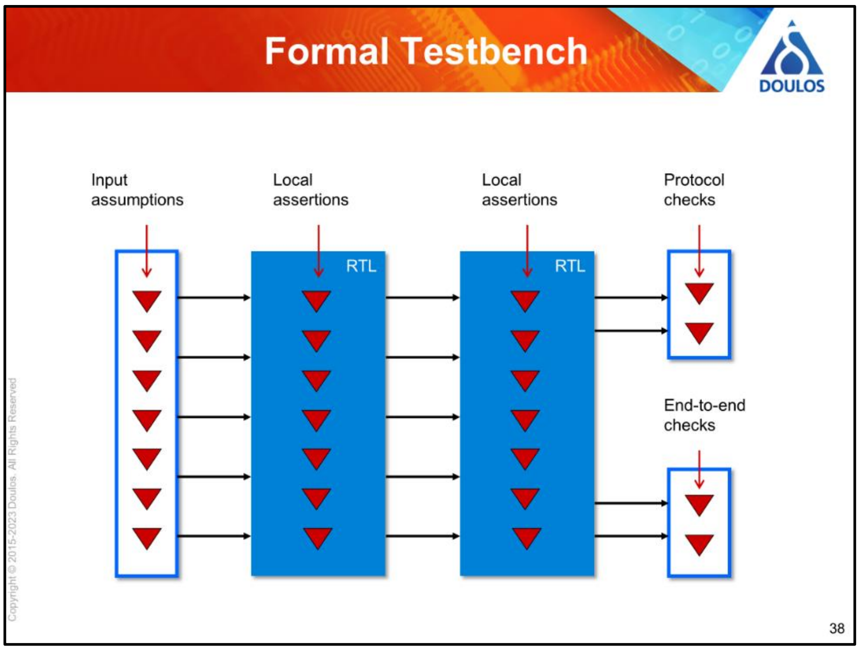
- Formal is not hard to use!

Give it your RTL code
Write some properties
Push the Go button
Wait for some results
Decide what to do next

Proof
Counter-example
Inconclusive

- The problem is understanding what formal can and cannot do

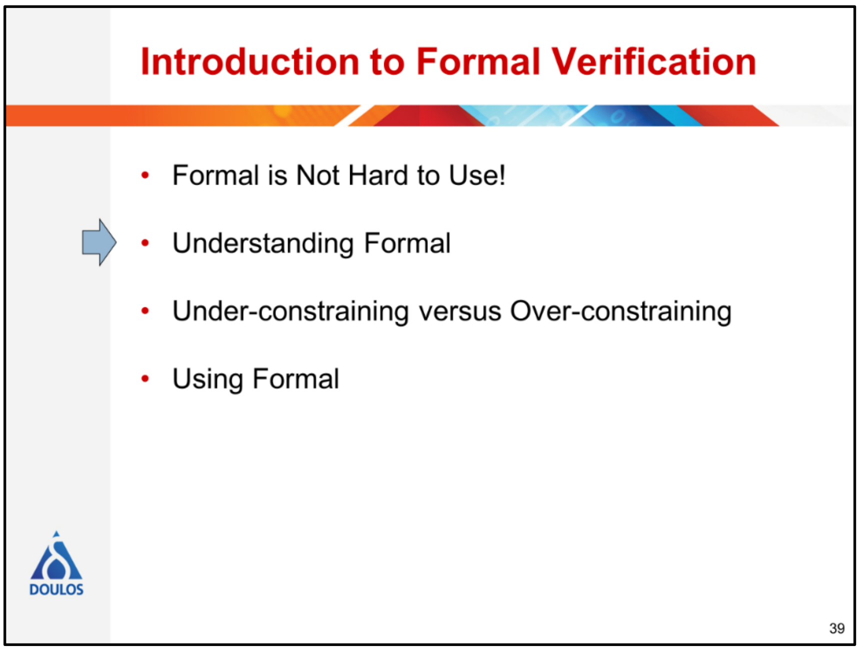
37



Formal Testbench

38

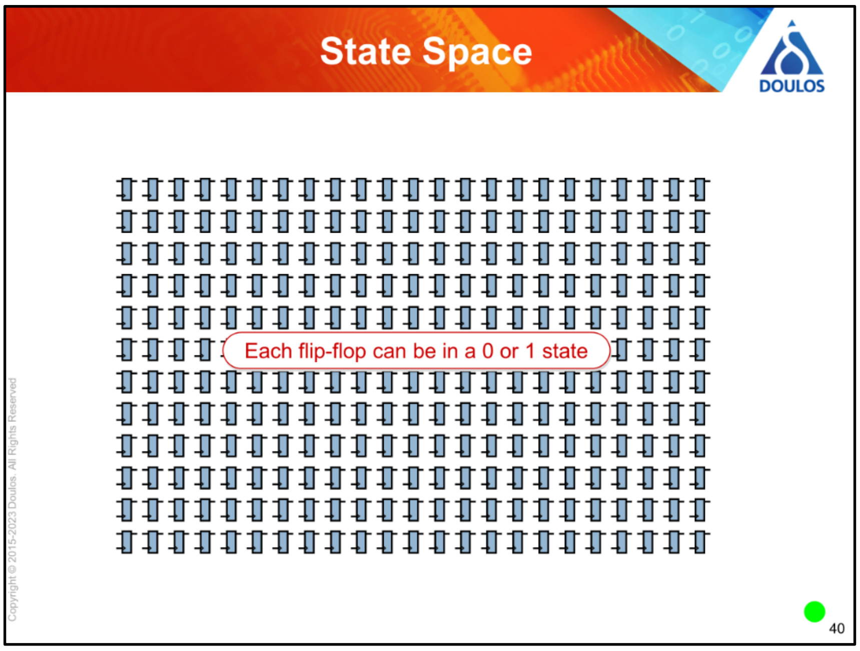
Understanding Formal



Introduction to Formal Verification

- Formal is Not Hard to Use!
- Understanding Formal
- Under-constraining versus Over-constraining
- Using Formal

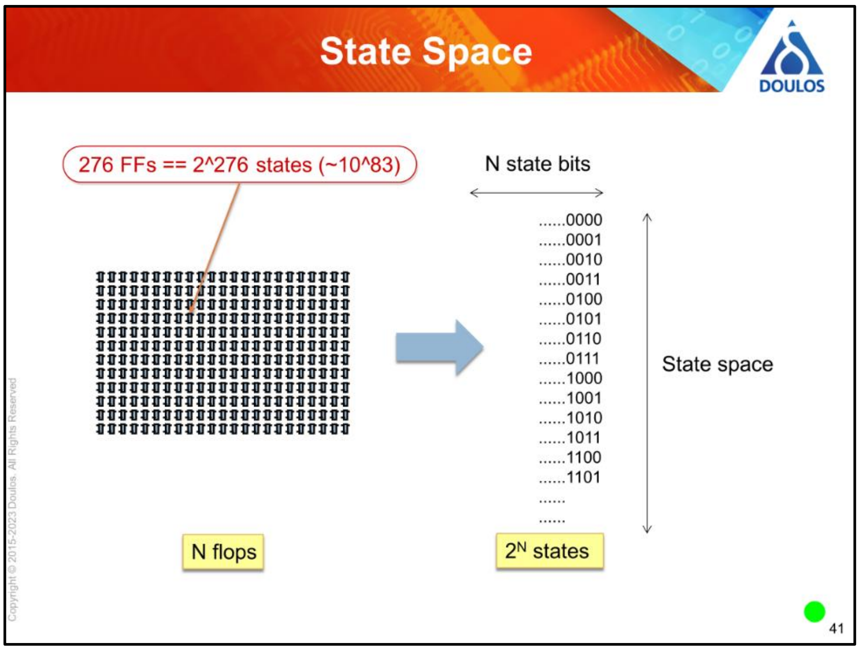
39



State Space

Each flip-flop can be in a 0 or 1 state

40



State Space

276 FFs == 2^276 states (~10^83)

N state bits

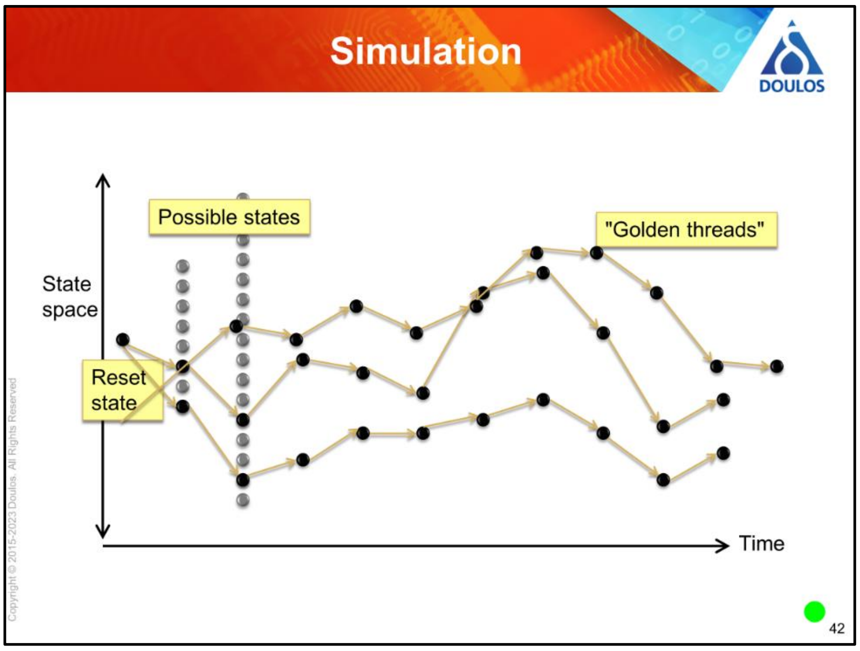
```
......0000
......0001
......0010
......0011
......0100
......0101
......0110
......0111
......1000
......1001
......1010
......1011
......1100
......1101
......
......
```

State space

N flops

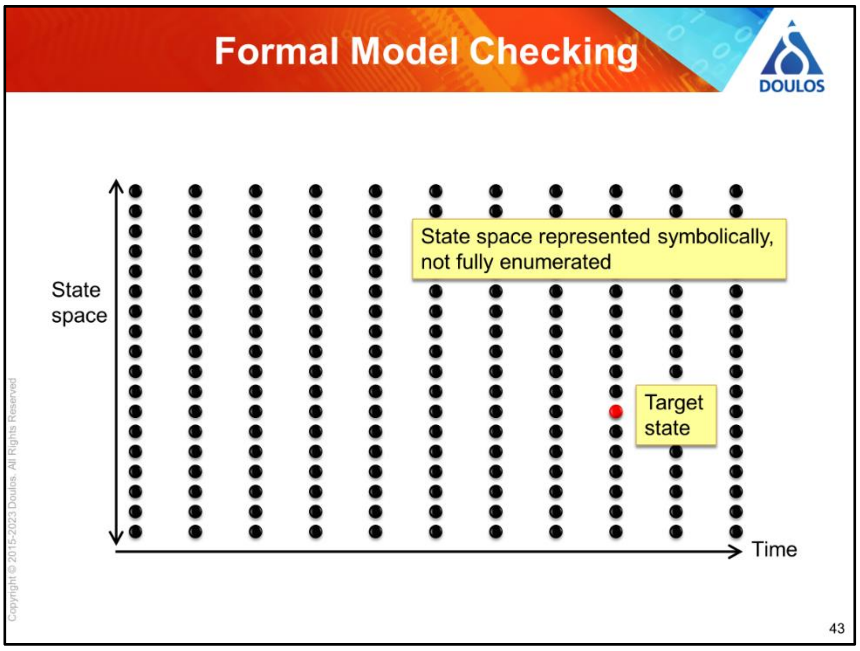
2^N states

41



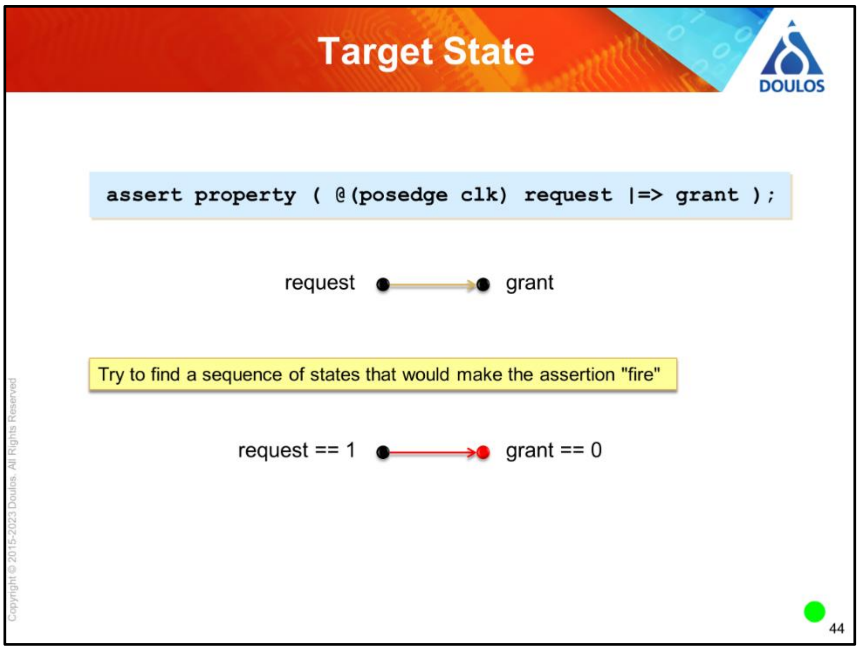
Simulation

42



Formal Model Checking

43



Target State

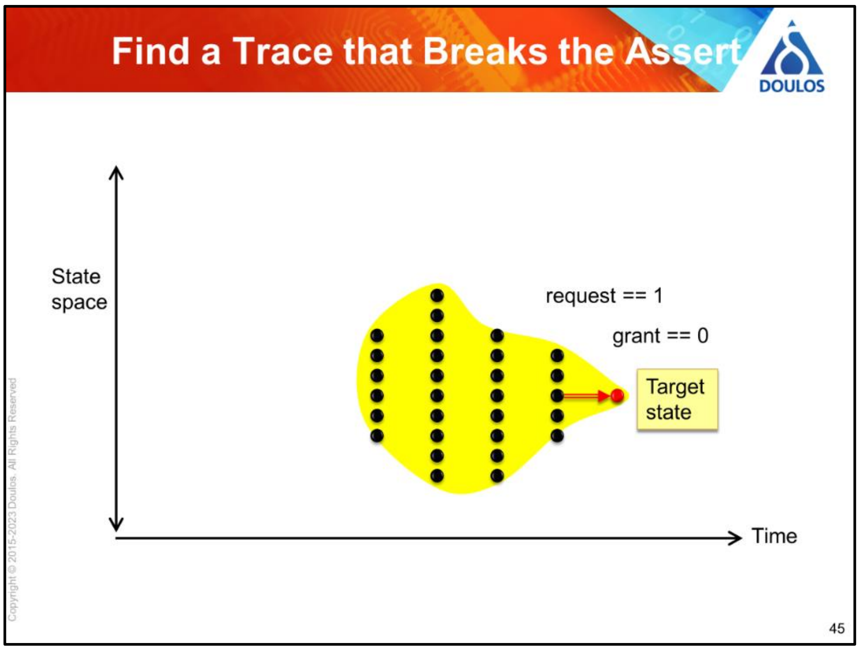
```
assert property ( @(posedge clk) request |=> grant );
```

request
grant

Try to find a sequence of states that would make the assertion "fire"

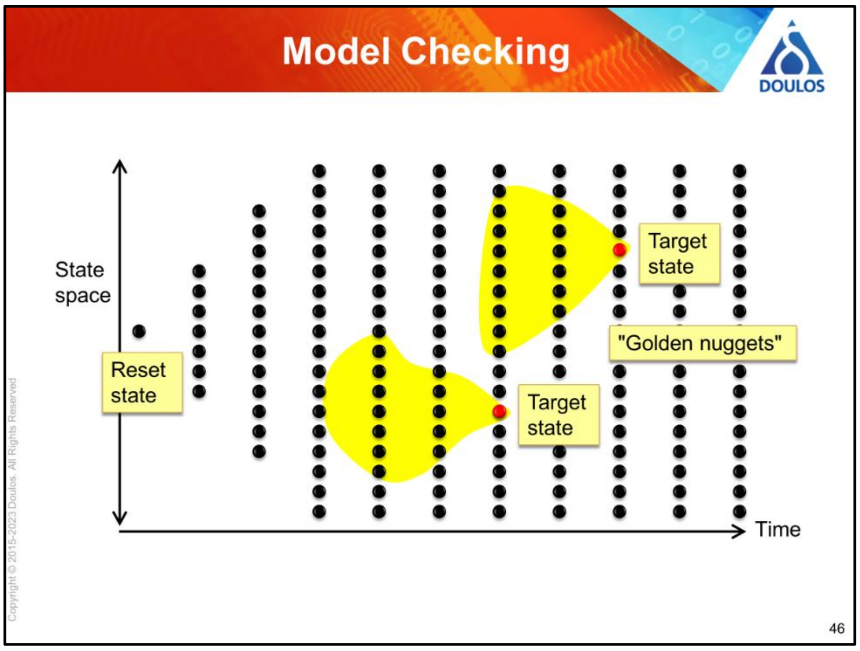
request == 1
grant == 0

44



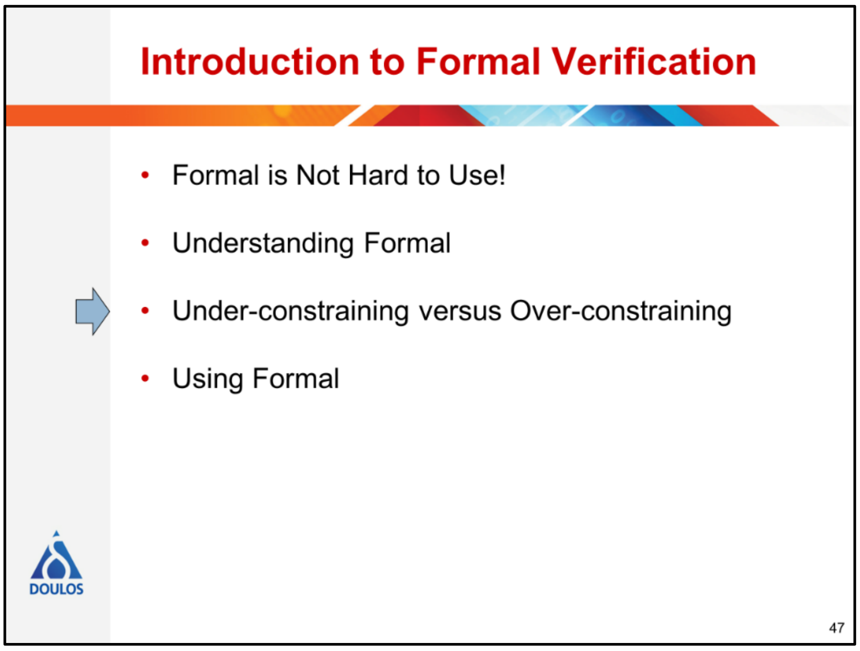
Find a Trace that Breaks the Assert

45


Model Checking

46

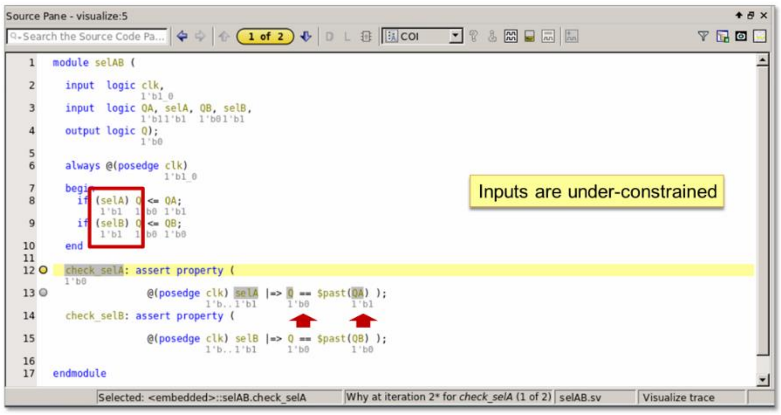
Under-constraining versus Over-constraining


Introduction to Formal Verification

- Formal is Not Hard to Use!
- Understanding Formal
- Under-constraining versus Over-constraining
- Using Formal

47

The Failing Example



Source Pane - visualize:5

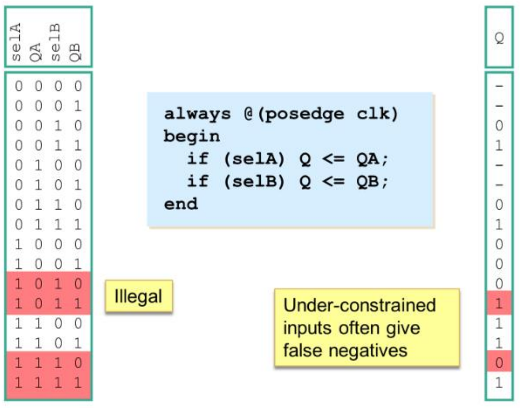
```
module selAB (
  input  logic clk,
               1'b1_0
  input  logic QA, selA, QB, selB,
               1'b11'b1  1'b01'b1
  output logic Q);
               1'b0

  always @(posedge clk)
                   1'b1_0
  begin
    if (selA) Q <= QA;
       1'b1   1'b0 1'b1
    if (selB) Q <= QB;
       1'b1   1'b0 1'b0
  end

  check_selA: assert property (
  1'b0
                @(posedge clk) selA |=> Q == $past(QA) );
                        1'b..1'b1   1'b0       1'b1
  check_selB: assert property (

                @(posedge clk) selB |=> Q == $past(QB) );
                        1'b..1'b1   1'b0       1'b0

endmodule
```

Selected: <embedded>::selAB.check_selA
Why at iteration 2* for *check_selA* (1 of 2)
selAB.sv
Visualize trace

Inputs are under-constrained

48

Under-constrained Inputs



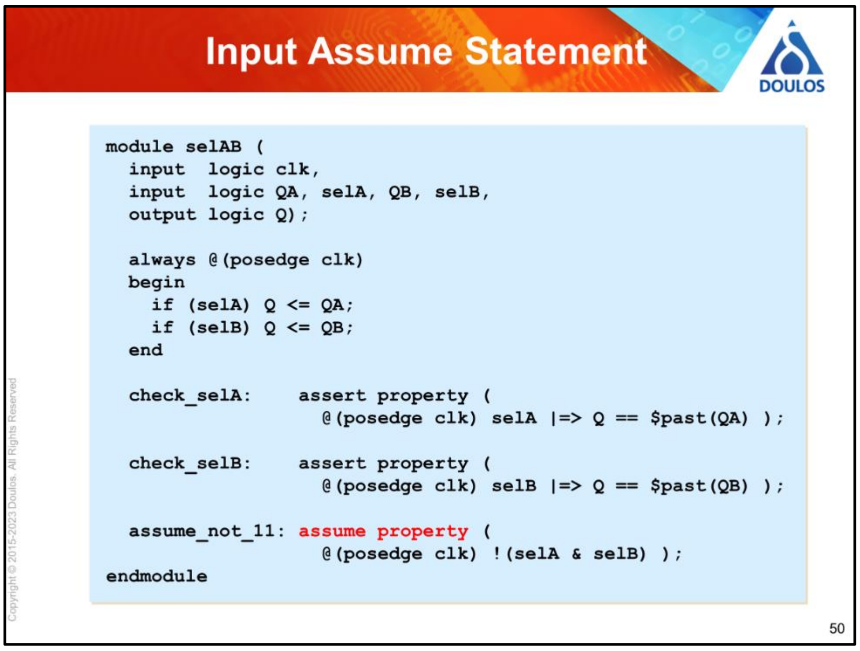
selA	QA	selB	QB	Q
0	0	0	0	-
0	0	0	1	-
0	0	1	0	0
0	0	1	1	1
0	1	0	0	-
0	1	0	1	-
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Illegal

```
always @(posedge clk)
begin
  if (selA) Q <= QA;
  if (selB) Q <= QB;
end
```

Under-constrained inputs often give false negatives

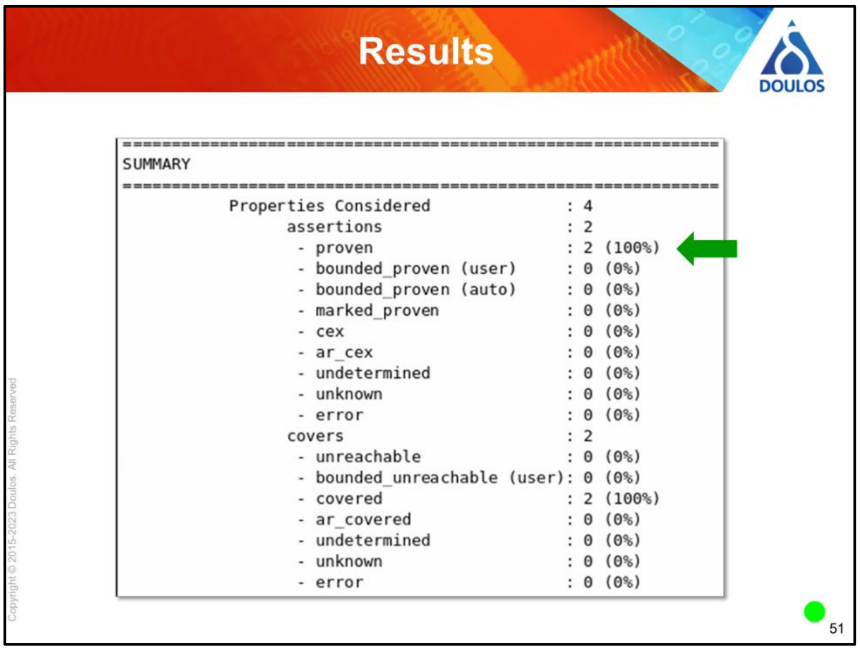
49



Input Assume Statement

```
module selAB (
  input  logic clk,
  input  logic QA, selA, QB, selB,
  output logic Q);

  always @(posedge clk)
  begin
    if (selA) Q <= QA;
    if (selB) Q <= QB;
  end

  check_selA:    assert property (
                   @(posedge clk) selA |=> Q == $past(QA) );

  check_selB:    assert property (
                   @(posedge clk) selB |=> Q == $past(QB) );

  assume_not_11: assume property (
                   @(posedge clk) !(selA & selB) );
endmodule
```

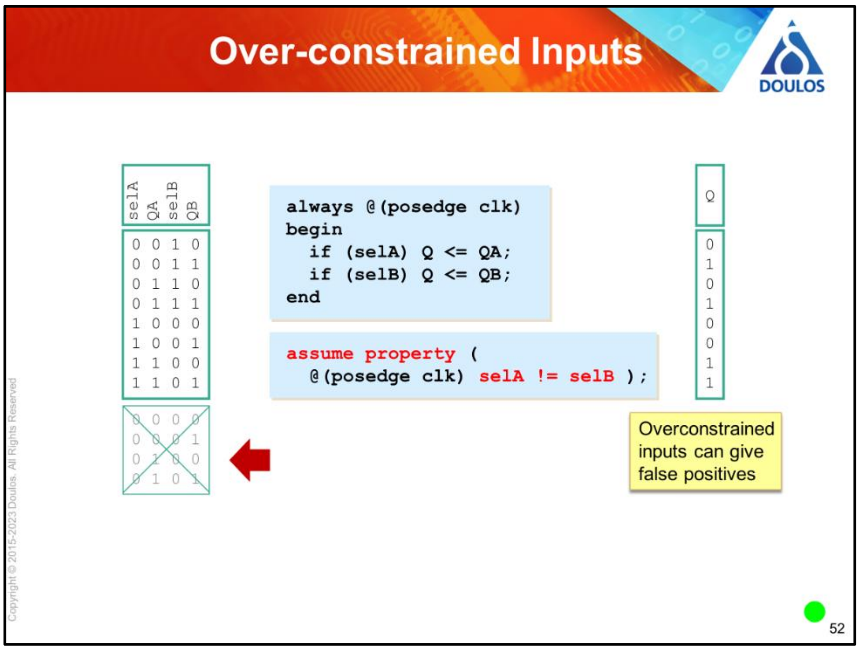
50



Results

```
===============================================================
SUMMARY
===============================================================
           Properties Considered              : 4
                 assertions                   : 2
                  - proven                    : 2 (100%)
                  - bounded_proven (user)     : 0 (0%)
                  - bounded_proven (auto)     : 0 (0%)
                  - marked_proven             : 0 (0%)
                  - cex                       : 0 (0%)
                  - ar_cex                    : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
                 covers                       : 2
                  - unreachable               : 0 (0%)
                  - bounded_unreachable (user): 0 (0%)
                  - covered                   : 2 (100%)
                  - ar_covered                : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
```

51



Over-constrained Inputs

selA	QA	selB	QB	Q
0	0	1	0	0
0	0	1	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	1	0	0	1
1	1	0	1	1

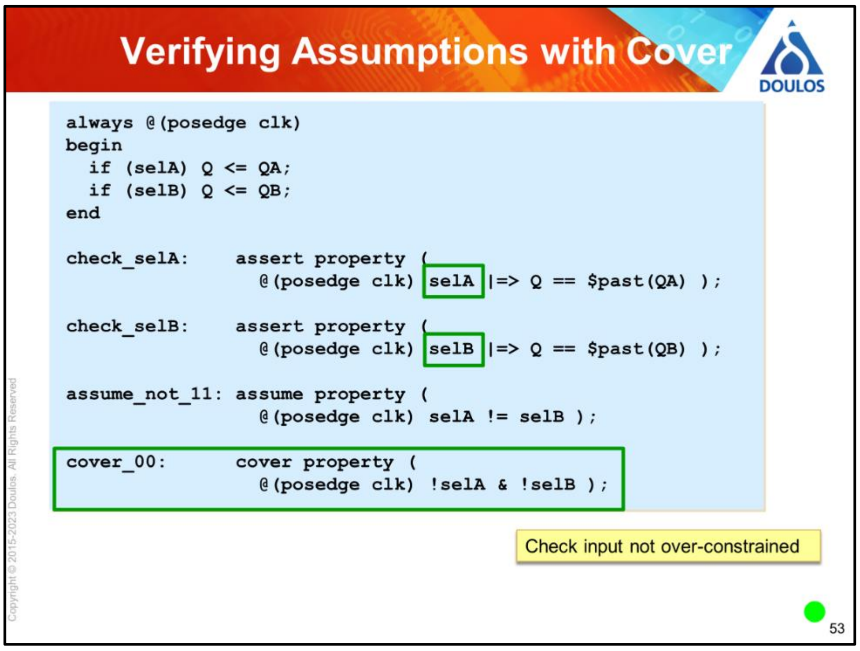
~~0~~	~~0~~	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~	~~1~~
~~0~~	~~1~~	~~0~~	~~0~~
~~0~~	~~1~~	~~0~~	~~1~~

```
always @(posedge clk)
begin
  if (selA) Q <= QA;
  if (selB) Q <= QB;
end
```

```
assume property (
  @(posedge clk) selA != selB );
```

Overconstrained inputs can give false positives

52

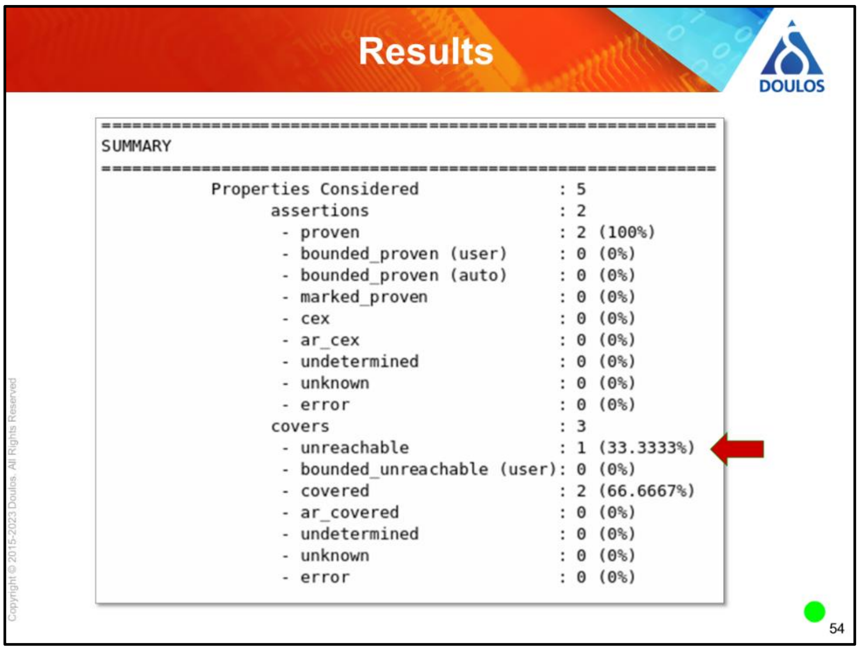


Verifying Assumptions with Cover

```
always @(posedge clk)
begin
  if (selA) Q <= QA;
  if (selB) Q <= QB;
end

check_selA:     assert property (
                  @(posedge clk) selA |=> Q == $past(QA) );

check_selB:     assert property (
                  @(posedge clk) selB |=> Q == $past(QB) );

assume_not_11: assume property (
                  @(posedge clk) selA != selB );

cover_00:       cover property (
                  @(posedge clk) !selA & !selB );
```

Check input not over-constrained

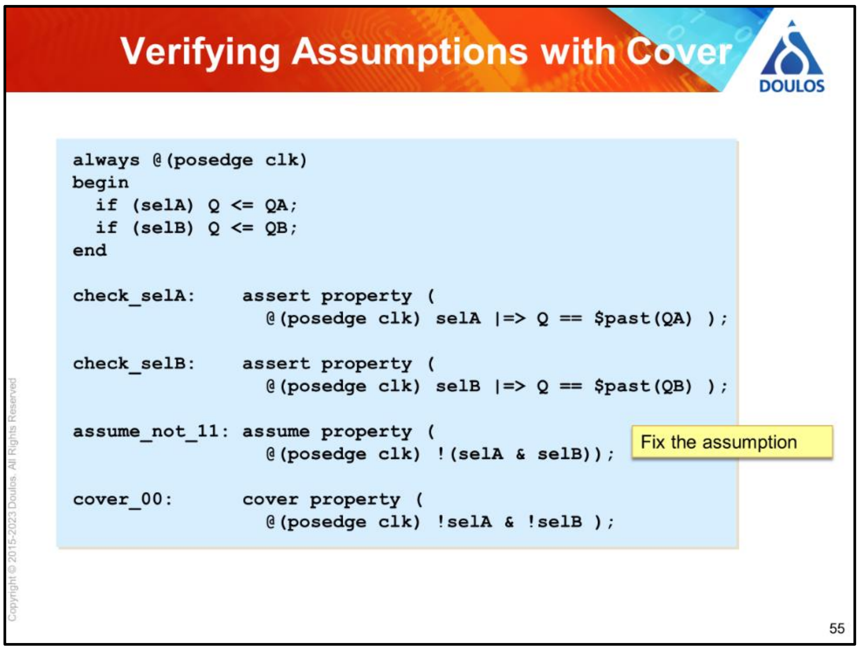
53



Results

```
==============================================================
SUMMARY
==============================================================
           Properties Considered              : 5
                 assertions                   : 2
                  - proven                    : 2 (100%)
                  - bounded_proven (user)     : 0 (0%)
                  - bounded_proven (auto)     : 0 (0%)
                  - marked_proven             : 0 (0%)
                  - cex                       : 0 (0%)
                  - ar_cex                    : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
                 covers                       : 3
                  - unreachable               : 1 (33.3333%)
                  - bounded_unreachable (user): 0 (0%)
                  - covered                   : 2 (66.6667%)
                  - ar_covered                : 0 (0%)
                  - undetermined              : 0 (0%)
                  - unknown                   : 0 (0%)
                  - error                     : 0 (0%)
```

54

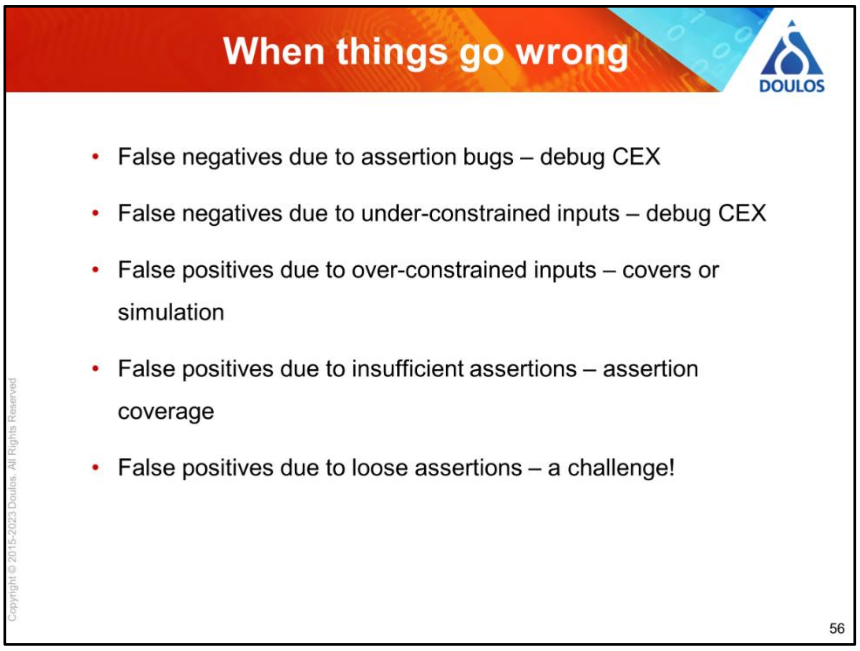


Verifying Assumptions with Cover

```
always @(posedge clk)
begin
  if (selA) Q <= QA;
  if (selB) Q <= QB;
end

check_selA:    assert property (
                 @(posedge clk) selA |=> Q == $past(QA) );

check_selB:    assert property (
                 @(posedge clk) selB |=> Q == $past(QB) );

assume_not_11: assume property (
                 @(posedge clk) !(selA & selB));

cover_00:      cover property (
                 @(posedge clk) !selA & !selB );
```

Fix the assumption

55

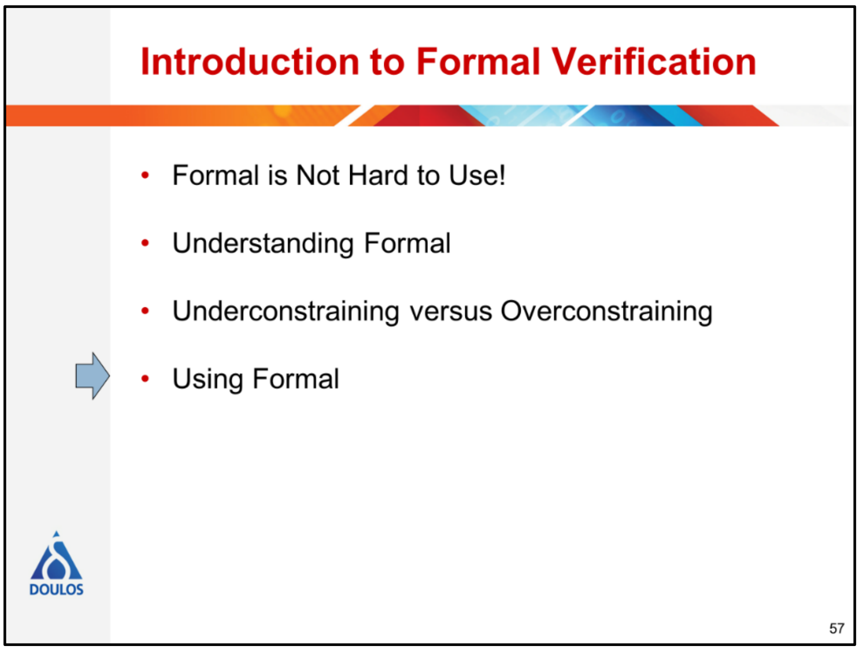


When things go wrong

- False negatives due to assertion bugs – debug CEX
- False negatives due to under-constrained inputs – debug CEX
- False positives due to over-constrained inputs – covers or simulation
- False positives due to insufficient assertions – assertion coverage
- False positives due to loose assertions – a challenge!

56

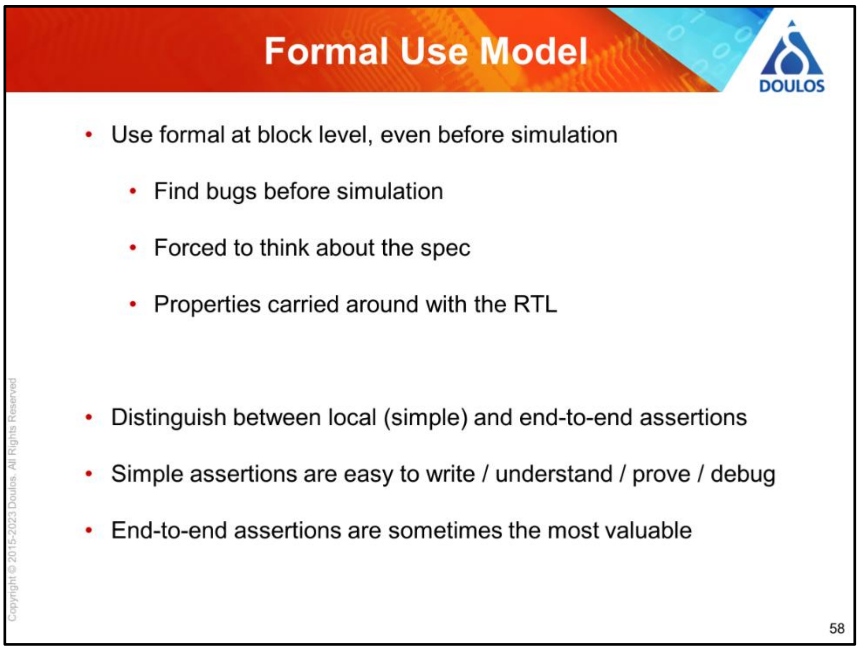
Using Formal



Introduction to Formal Verification

- Formal is Not Hard to Use!
- Understanding Formal
- Underconstraining versus Overconstraining
- Using Formal

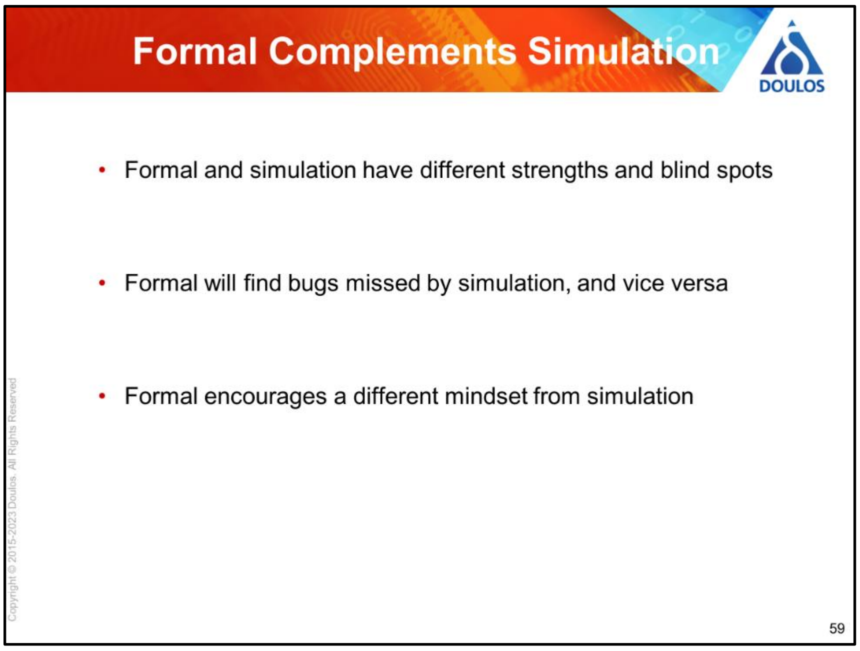
57



Formal Use Model

- Use formal at block level, even before simulation
  - Find bugs before simulation
  - Forced to think about the spec
  - Properties carried around with the RTL
- Distinguish between local (simple) and end-to-end assertions
- Simple assertions are easy to write / understand / prove / debug
- End-to-end assertions are sometimes the most valuable

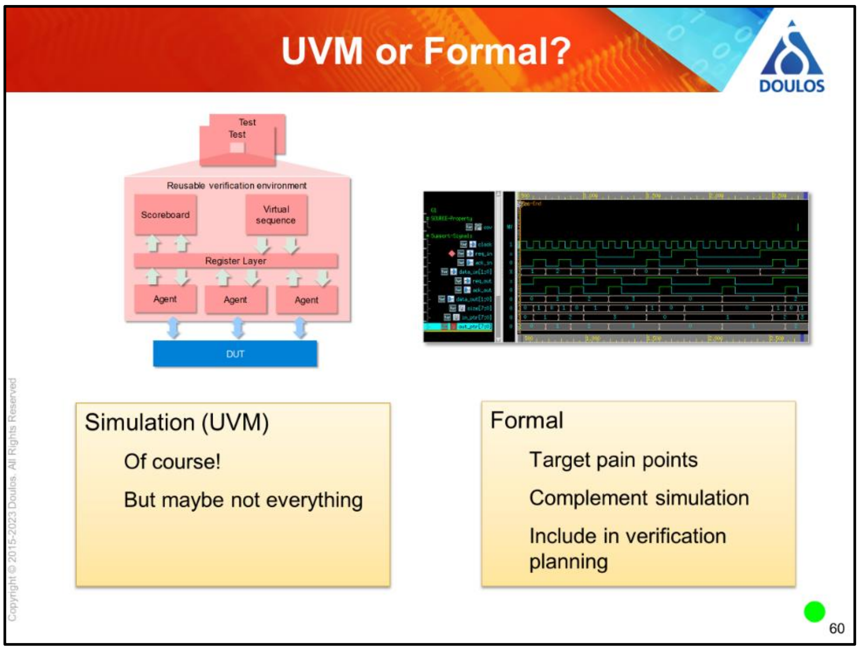
58



Formal Complements Simulation

- Formal and simulation have different strengths and blind spots
- Formal will find bugs missed by simulation, and vice versa
- Formal encourages a different mindset from simulation

59


Test
Test
Reusable verification environment
Scoreboard
Virtual sequence
Register Layer
Agent
Agent
Agent
DUT

UVM or Formal?

DOULOS

Simulation (UVM)

- Of course!
- But maybe not everything

Formal

- Target pain points
- Complement simulation
- Include in verification planning

60

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » AMD » Intel FPGA
Embedded Software	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Android » Arm » Rust » Zephyr
Python, Edge AI & Machine Learning	AI & DEEP LEARNING

SystemVerilog OUT OF THE BOX
UVM
SYSTEMC
yocto PROJECT PARTICIPANT
Zephyr
freeRTOS
SECURE EMBEDDED
arm APPROVED TRAINING PARTNER

DOULOS

www.doulos.com