Norwegian University of Science and Technology
Department of Electronics and Telecommunication

TFE4171 Design of Digital Systems 2

Exercise sets 3 and 4

Formal Verification of Digital Systems

Lab Tutorial

Dominik Stoffel
Wolfgang Kunz
TU Kaiserslautern, Germany

Overview

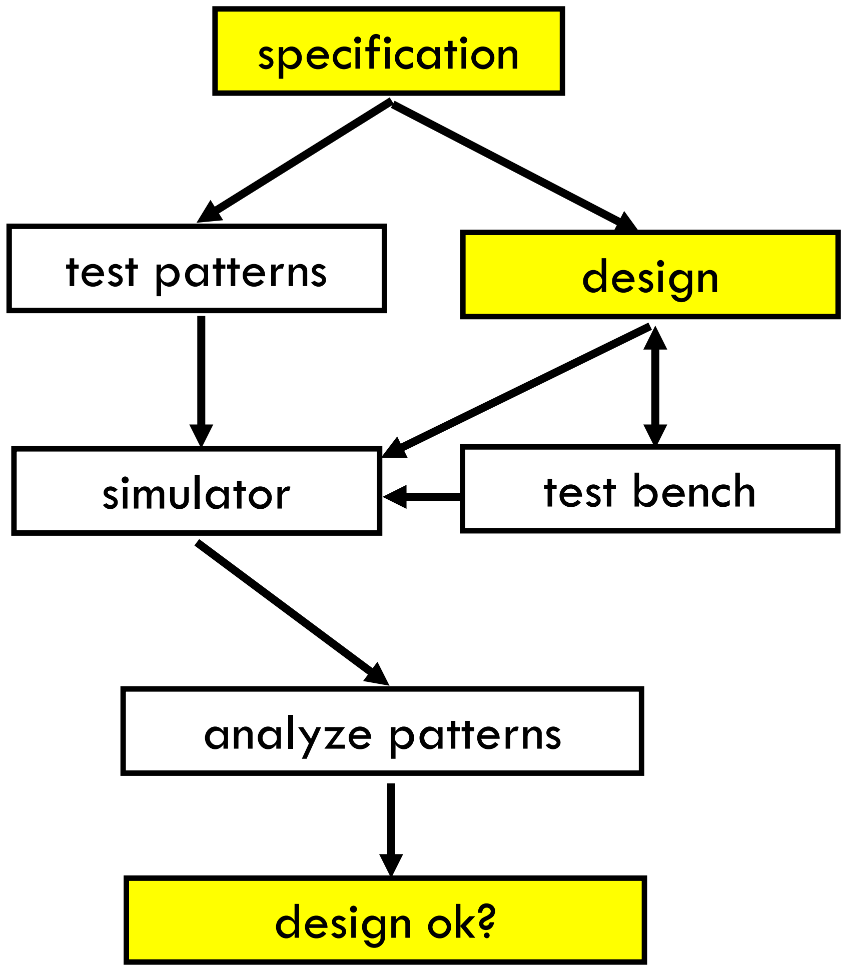
- OneSpin 360MV
  - GUI
  - Setup
  - Module Verification

- SystemVerilog Assertions (SVA) summary

Classical Verification Approach

Simulation

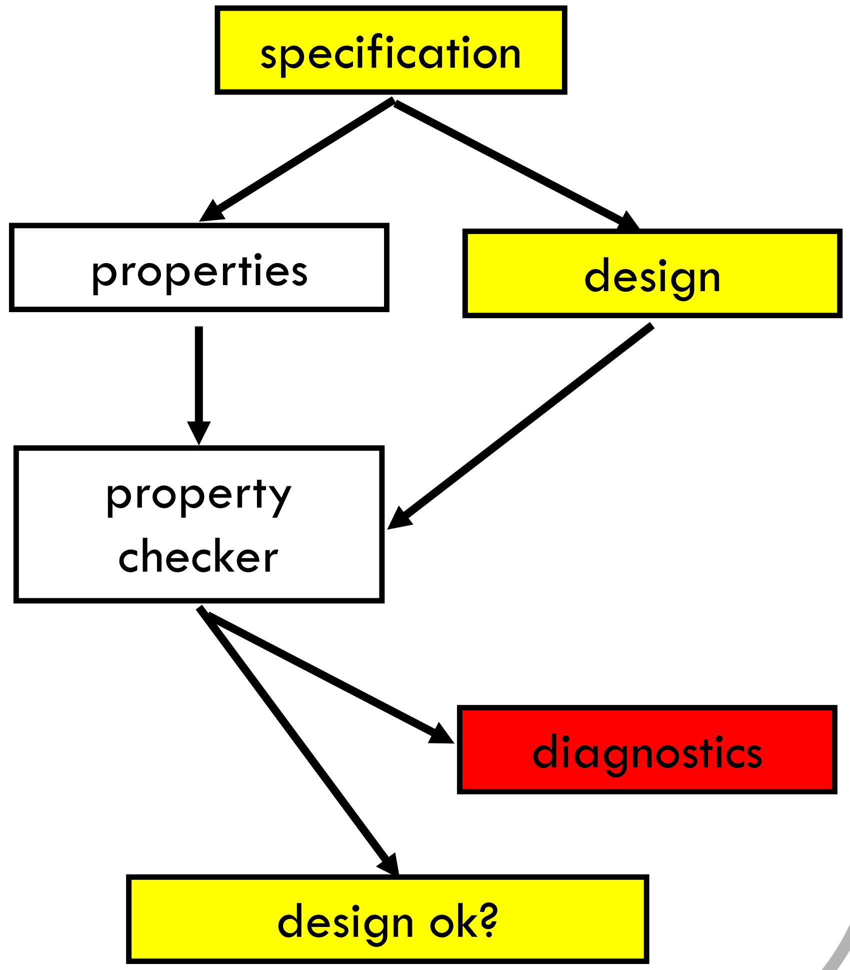
- set up a test bench
- create test patterns manually or write a pattern generator
- pattern analysis is not fully automated
- one can examine only a small number of patterns
- runtime limitations



Formal Verification

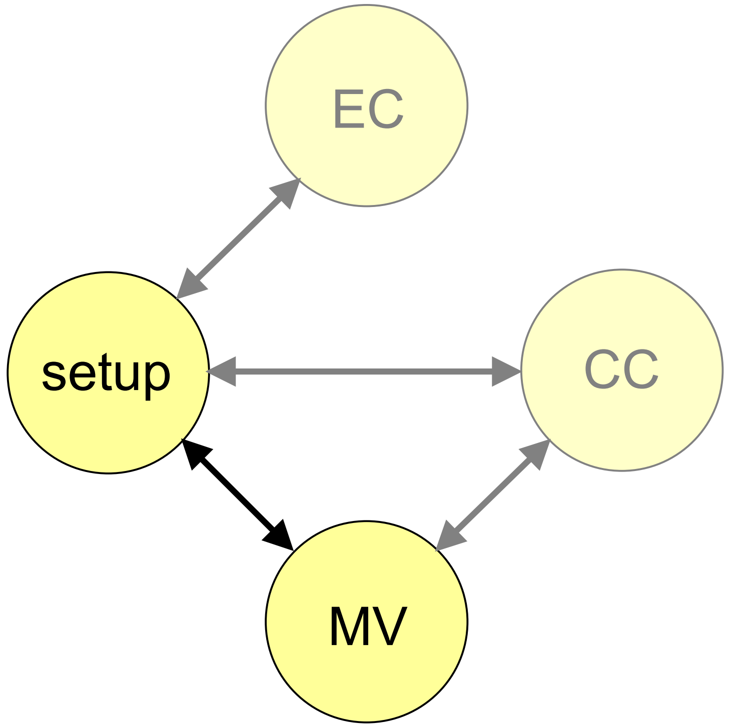
Property Checking

- no testbench needed
- create a set of properties
- diagnostic output is generated automatically in case of a failing property
- same as automatic, exhaustive simulation
- much faster than exhaustive simulation



Supported Modes

- setup mode
- module verification (MV)
- consistency check (CC)
- equivalence checking (EC)



Setup Mode

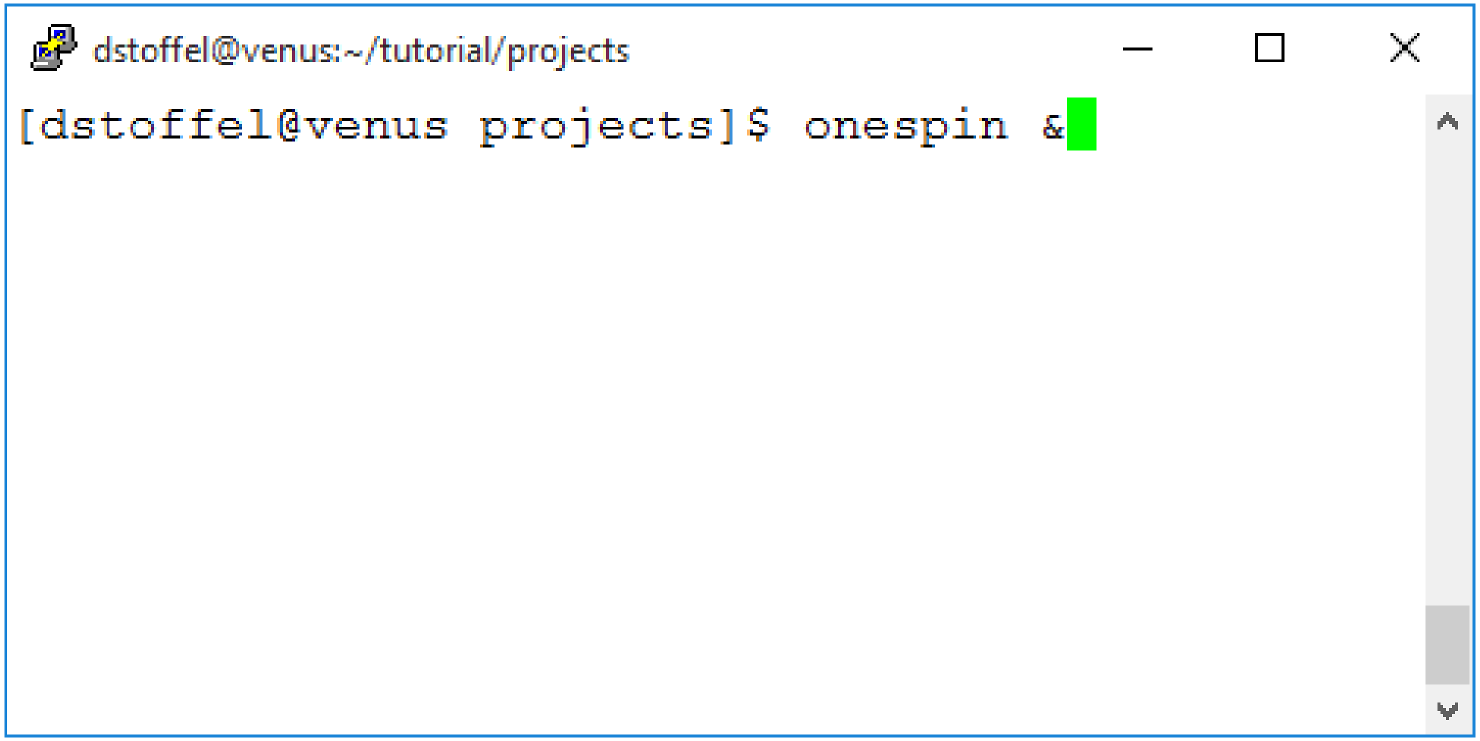
- `onespin` always starts in the setup mode

- load a VHDL design
- elaborate the design

- switch to the other modes for property checking (MV) or consistency checking (CC)

Getting Started

- Start the GUI by typing „`onespin`“ in a command shell.



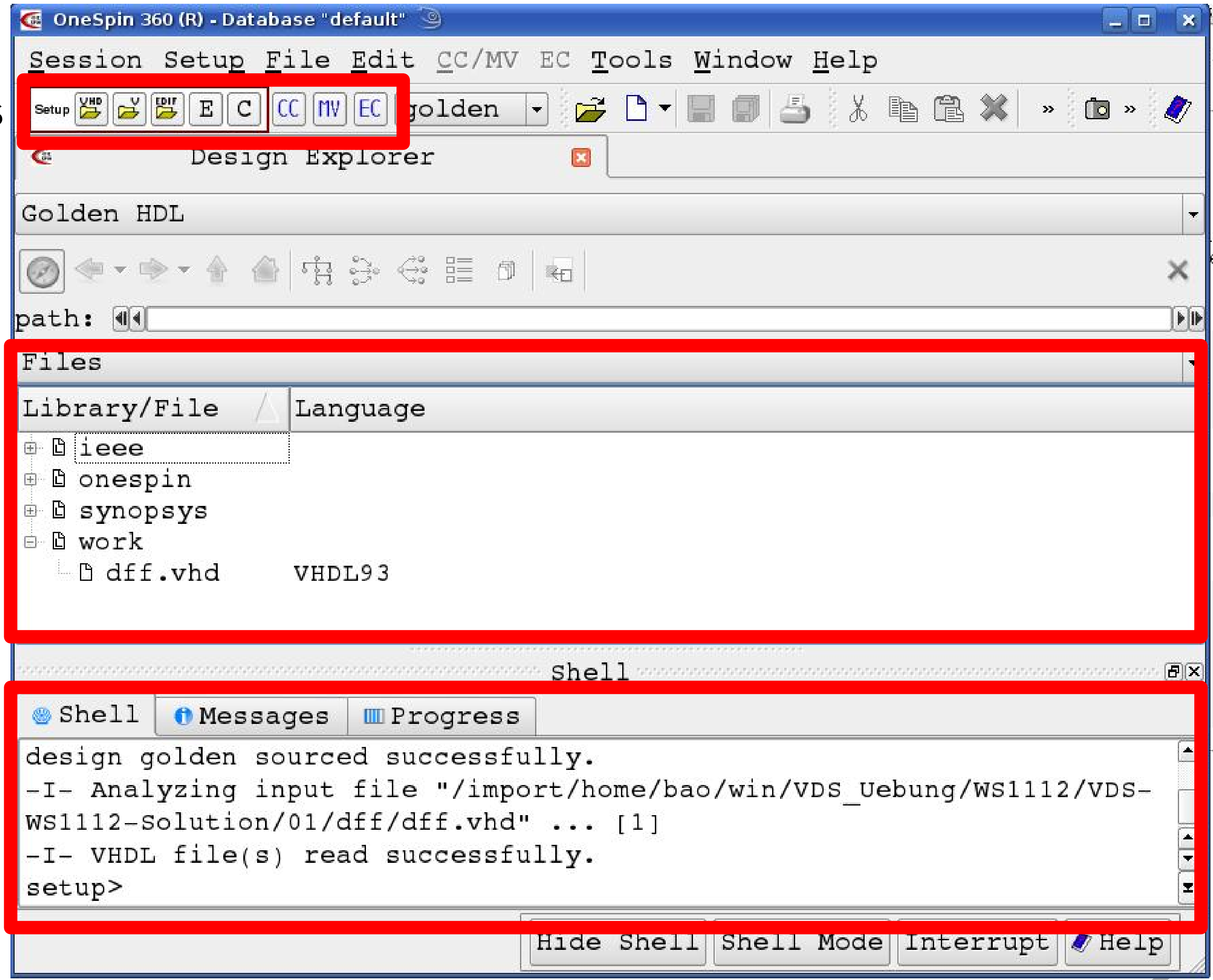
```
[dstoffel@venus projects]$ onespin &
```

Setup Mode

usage modes

source files

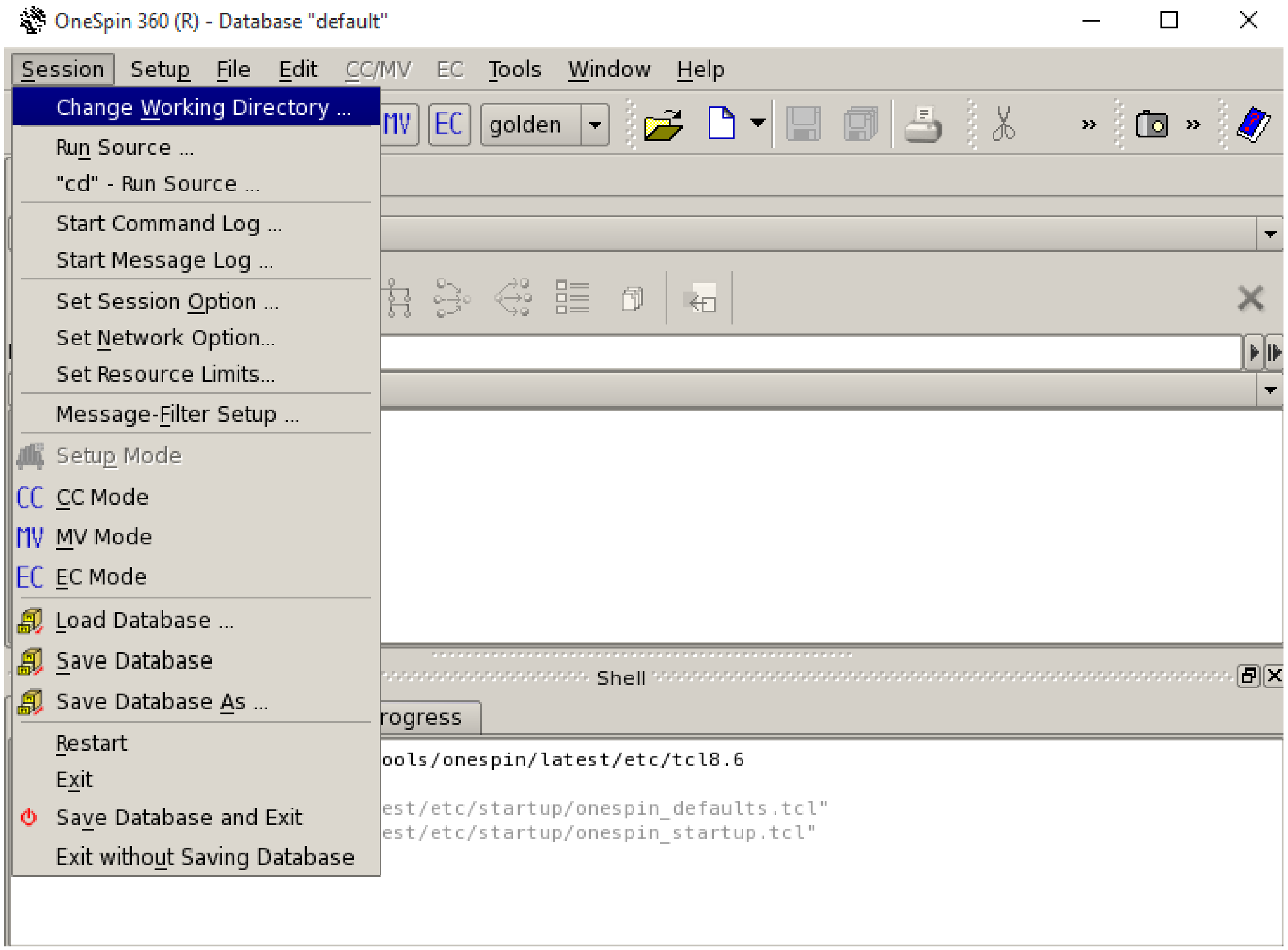
Tcl shell


OneSpin 360 (R) - Database "default"
Session Setup File Edit CC/MV EC Tools Window Help
Setup E C CC MV EC golden
Design Explorer
Golden HDL
path:
Files
Library/File Language
ieee
onespin
synopsys
work
dff.vhd VHDL93
Shell
Shell Messages Progress
Hide Shell Shell Mode Interrupt Help

```
design golden sourced successfully.
-I- Analyzing input file "/import/home/bao/win/VDS_Uebung/WS1112/VDS-WS1112-Solution/01/dff/dff.vhd" ... [1]
-I- VHDL file(s) read successfully.
setup>
```

Change Working Directory

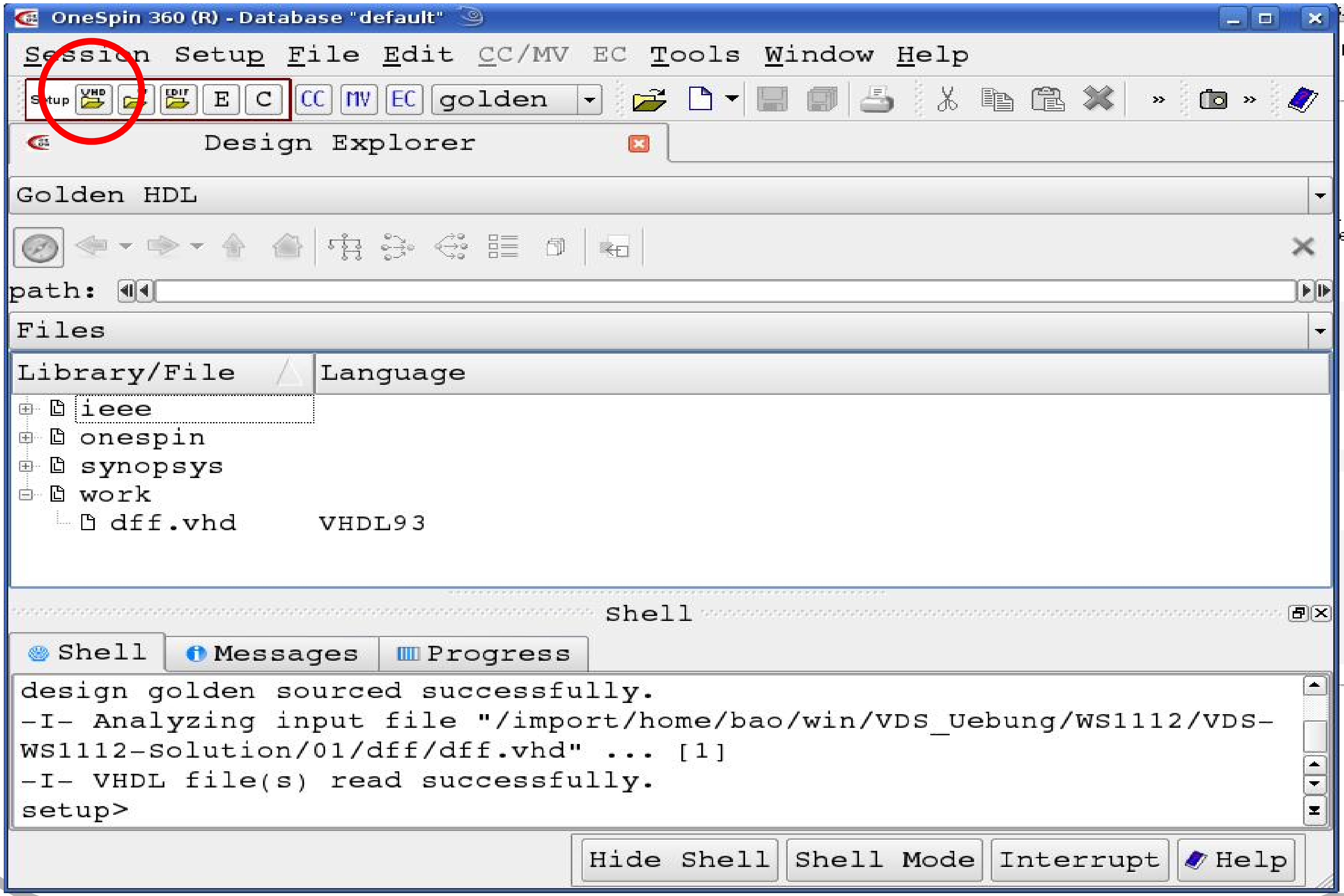
Change working directory to current project subdirectory, e.g., **projects/dff**.



```
ools/onespin/latest/etc/tcl8.6

est/etc/startup/onespin_defaults.tcl"
est/etc/startup/onespin_startup.tcl"
```

Setup Mode

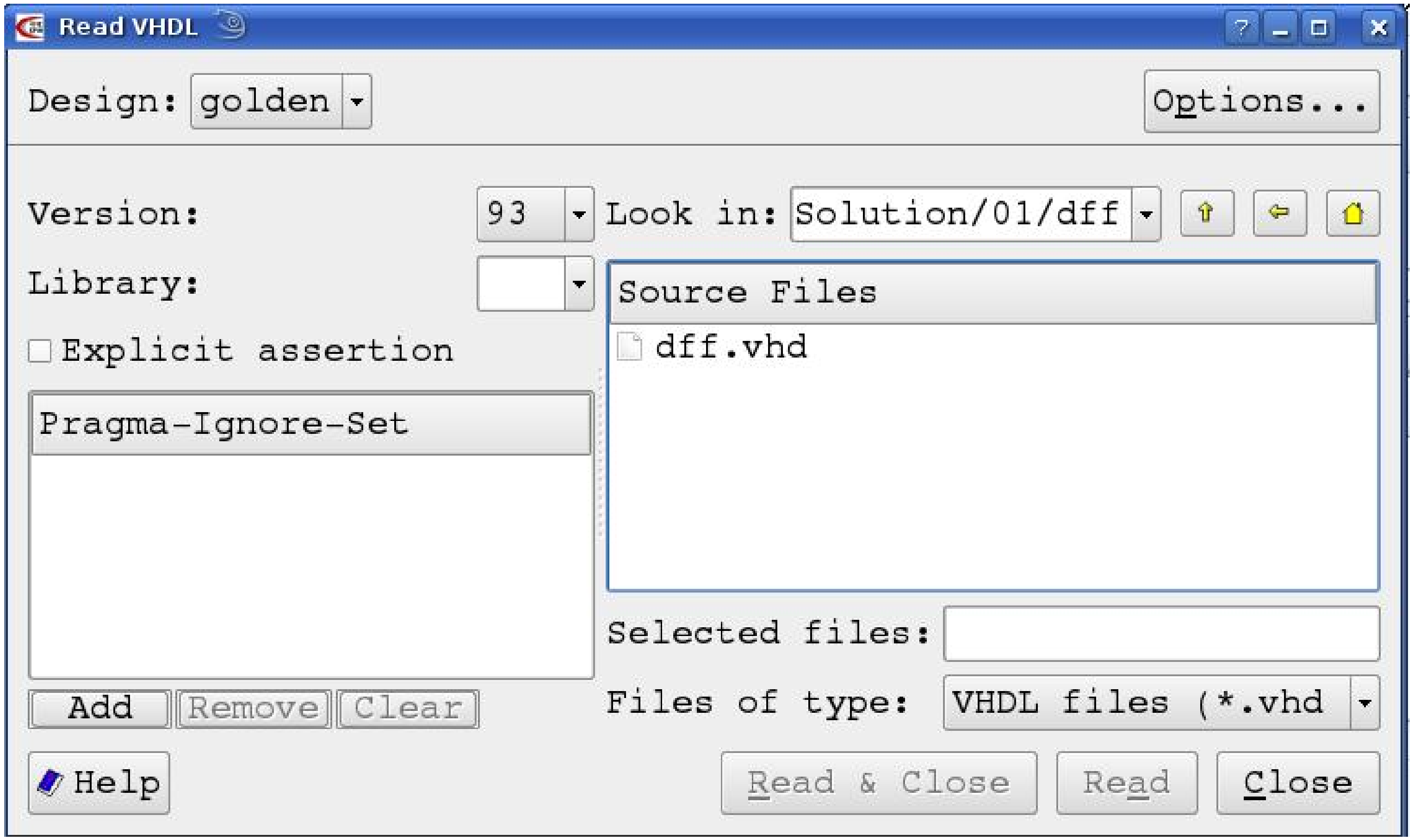
loading VHDL files (I):


OneSpin 360 (R) - Database "default"
Session Setup File Edit CC/MV EC Tools Window Help
Setup E C CC MV EC golden
Design Explorer
Golden HDL
path:
Files
Library/File Language
ieee
onespin
synopsys
work
dff.vhd VHDL93
Shell
Shell Messages Progress
Hide Shell Shell Mode Interrupt Help

```
design golden sourced successfully.
-I- Analyzing input file "/import/home/bao/win/VDS_Uebung/WS1112/VDS-
WS1112-Solution/01/dff/dff.vhd" ... [1]
-I- VHDL file(s) read successfully.
setup>
```

Setup Mode

reading VHDL files (II):


Read VHDL

Design: golden

Version: 93

Library:

☐ Explicit assertion

Look in: Solution/01/dff

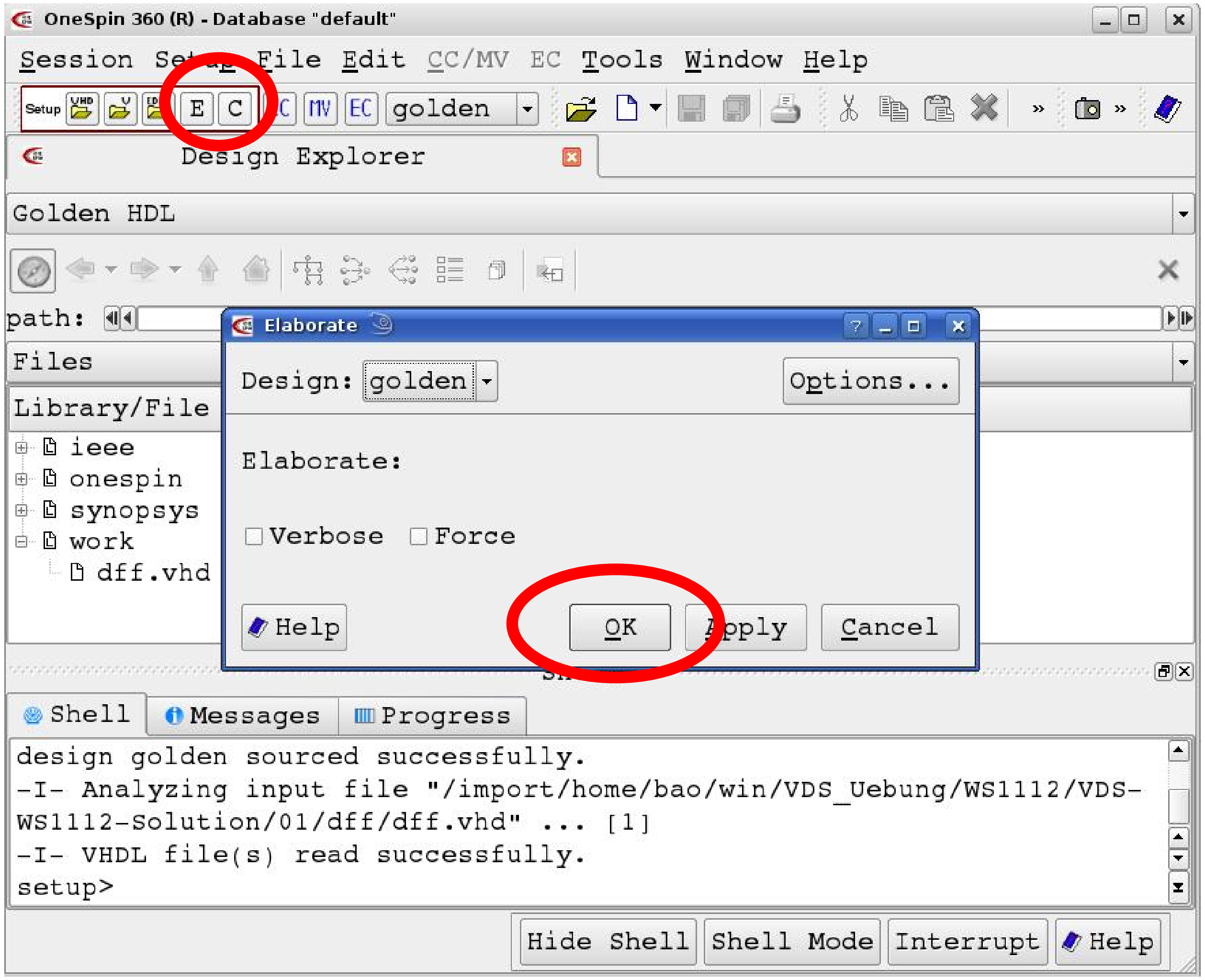
Source Files
dff.vhd

Selected files:

Files of type: VHDL files (*.vhd

Setup Mode

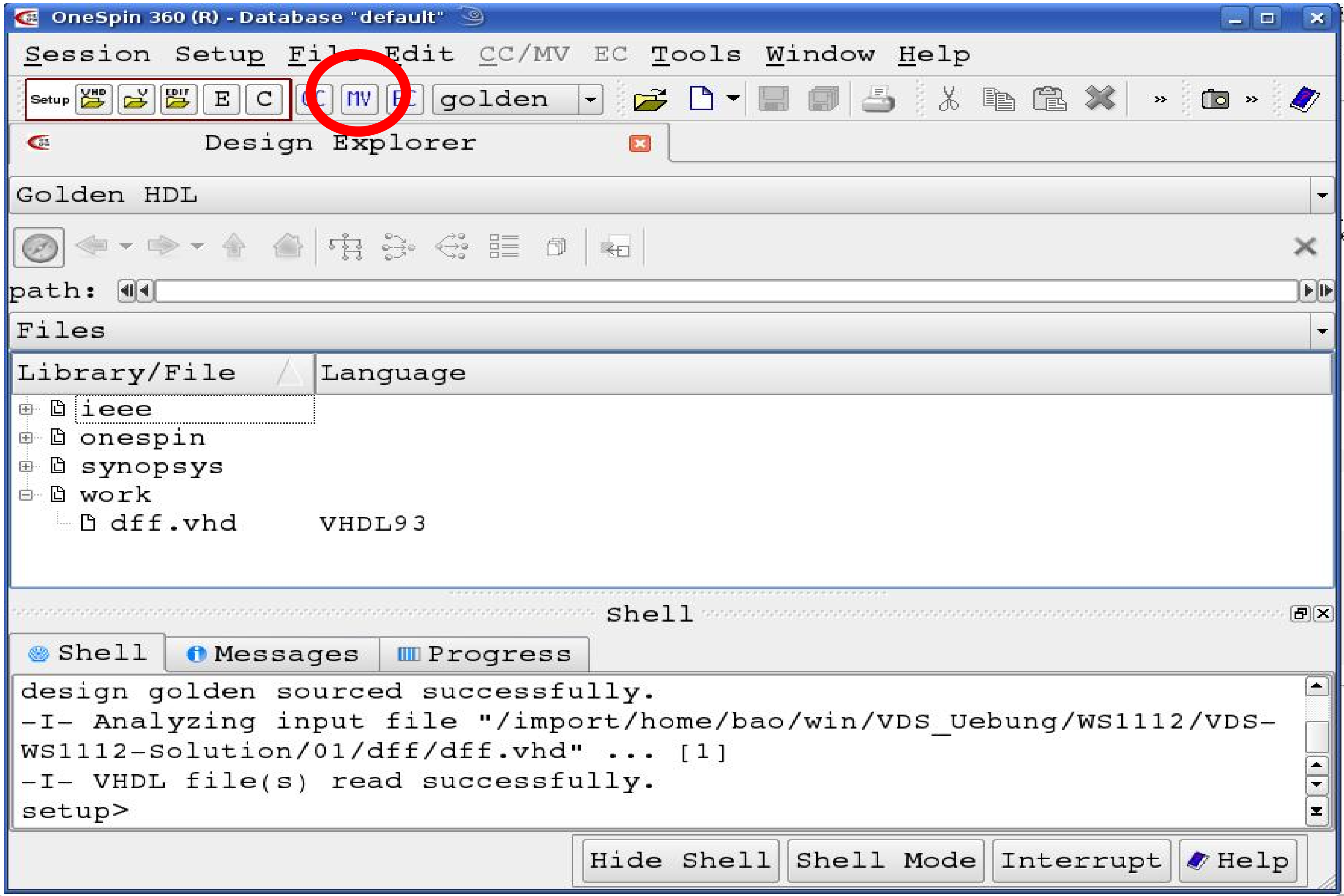
elaborate and compile the VHDL design



```
design golden sourced successfully.
-I- Analyzing input file "/import/home/bao/win/VDS_Uebung/WS1112/VDS-
WS1112-Solution/01/dff/dff.vhd" ... [1]
-I- VHDL file(s) read successfully.
setup>
```

Module Verification (MV)

- switch to MV mode:

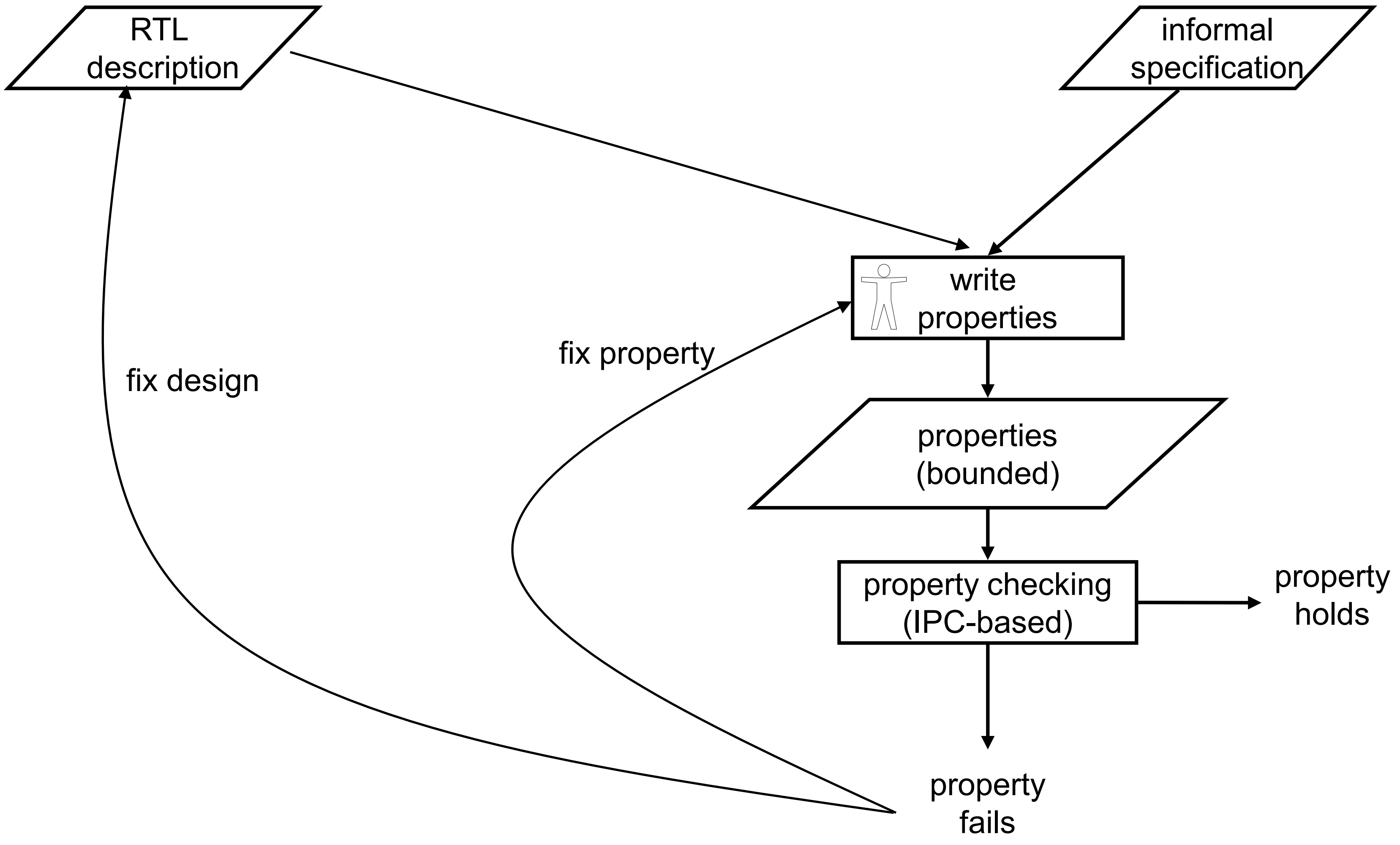

OneSpin 360 (R) - Database "default"
Session Setup File Edit CC/MV EC Tools Window Help
Setup E C CC MV EC golden
Design Explorer
Golden HDL
path:
Files
Library/File Language
ieee
onespin
synopsys
work
dff.vhd VHDL93
Shell
Shell Messages Progress
Hide Shell Shell Mode Interrupt Help

```
design golden sourced successfully.
-I- Analyzing input file "/import/home/bao/win/VDS_Uebung/WS1112/VDS-
WS1112-Solution/01/dff/dff.vhd" ... [1]
-I- VHDL file(s) read successfully.
setup>
```

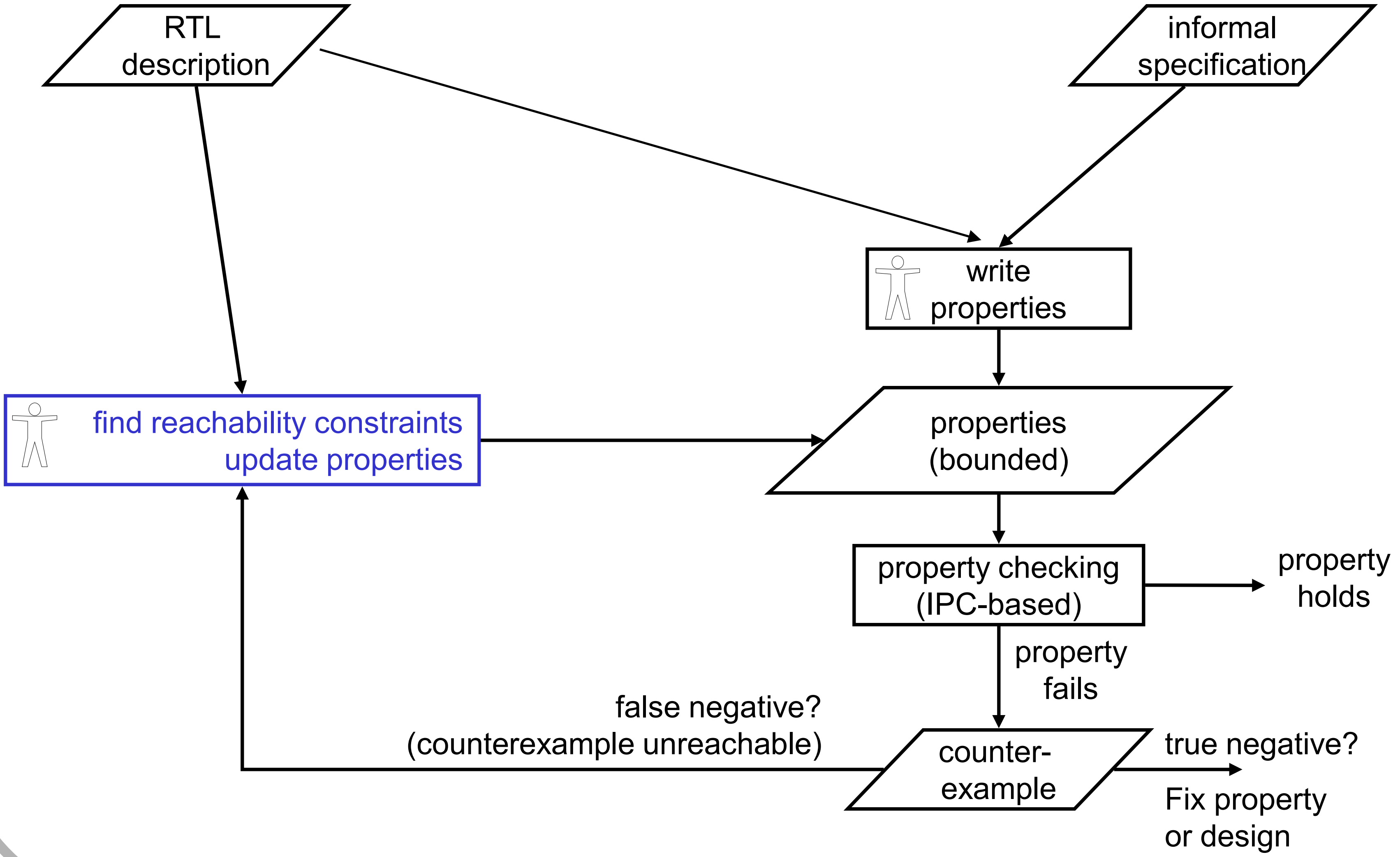
Module Verification (MV)

- Verify formally that a module implements a specified behavior.
- The behavior is described by a set of properties.
- Interval Property Checking (IPC)

Formal Verification – Flow



Unreachable Counterexamples

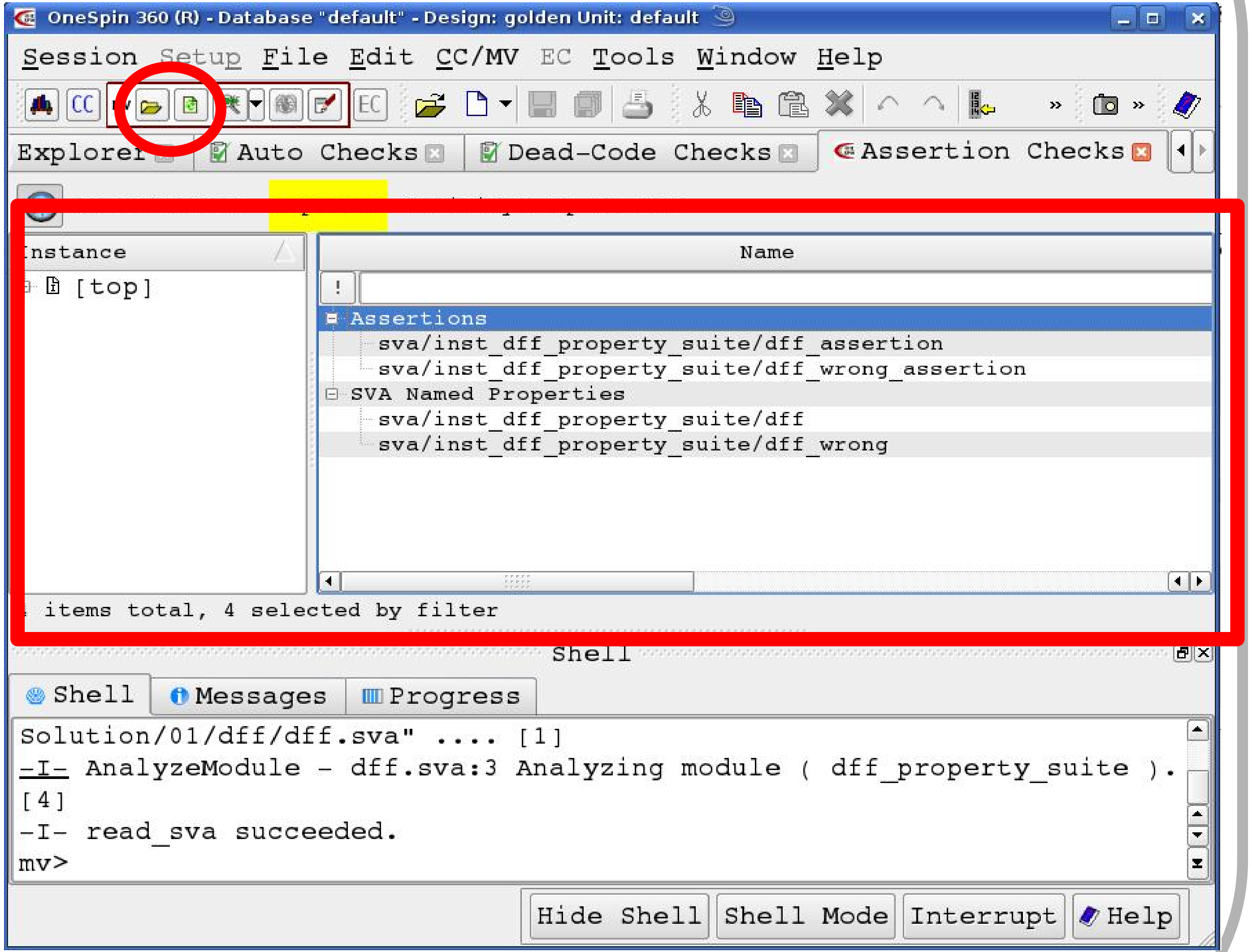


Module Verification (MV)

load assertions

Assertion list

warning and error messages are shown in the TCL shell



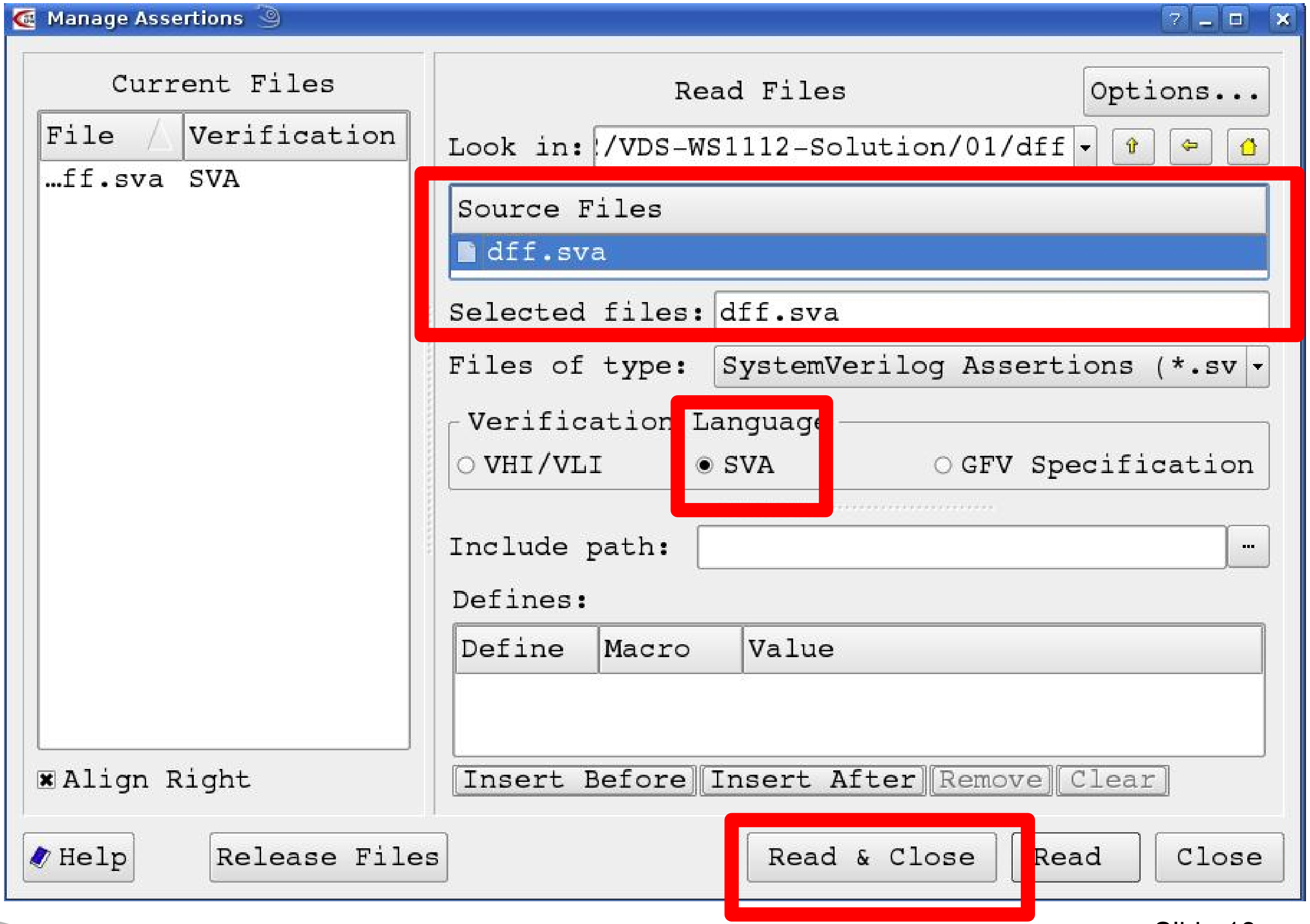
```
Assertions
    sva/inst_dff_property_suite/dff_assertion
    sva/inst_dff_property_suite/dff_wrong_assertion
SVA Named Properties
    sva/inst_dff_property_suite/dff
    sva/inst_dff_property_suite/dff_wrong
```

```
 items total, 4 selected by filter
```

```
Solution/01/dff/dff.sva" .... [1]
-I- AnalyzeModule - dff.sva:3 Analyzing module ( dff_property_suite ).
[4]
-I- read_sva succeeded.
mv>
```

Module Verification (MV)

load assertions



Manage Assertions

Current Files

File	Verification
…ff.sva	SVA

[✓ Align Right]

Read Files

Look in: !/VDS-WS1112-Solution/01/dff

Source Files
- dff.sva

Selected files: dff.sva

Files of type: SystemVerilog Assertions (*.sv

Verification Language: VHI/VLI [✓ SVA] GFV Specification

Include path:

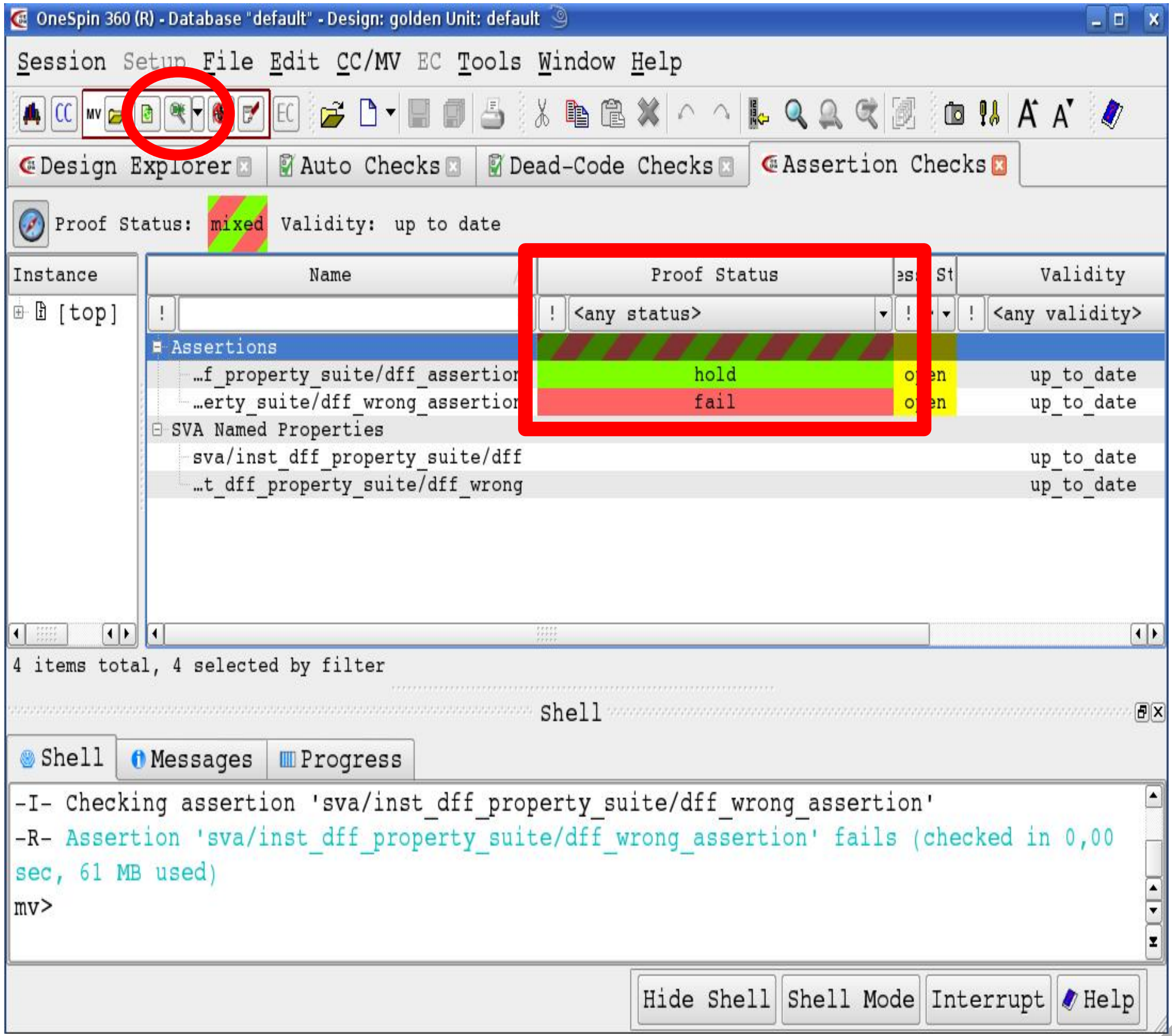
Defines:

Define	Macro	Value

Module Verification (MV)

prove assertions

An assertion can either hold or fail


OneSpin 360 (R) - Database "default" - Design: golden Unit: default
Session Setup File Edit CC/MV EC Tools Window Help
Design Explorer
Auto Checks
Dead-Code Checks
Assertion Checks
Shell
Messages
Progress
Hide Shell
Shell Mode
Interrupt
Help

Proof Status: mixed Validity: up to date

Instance	Name	Proof Status	St	Validity
[top]		<any status>		<any validity>
	Assertions			
	…f_property_suite/dff_assertion	hold	open	up_to_date
	…erty_suite/dff_wrong_assertion	fail	open	up_to_date
	SVA Named Properties			
	sva/inst_dff_property_suite/dff			up_to_date
	…t_dff_property_suite/dff_wrong			up_to_date

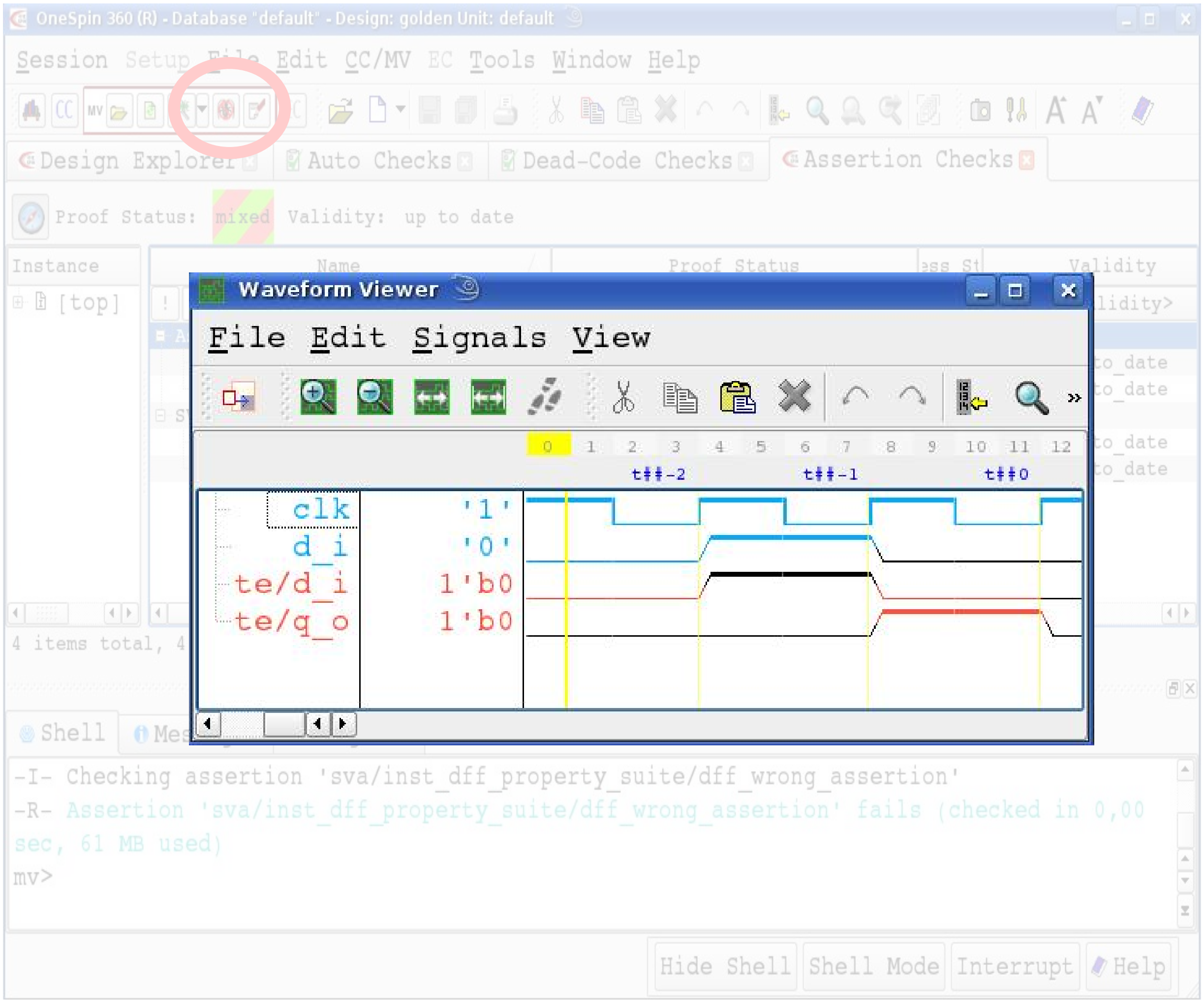
4 items total, 4 selected by filter

```
-I- Checking assertion 'sva/inst_dff_property_suite/dff_wrong_assertion'
-R- Assertion 'sva/inst_dff_property_suite/dff_wrong_assertion' fails (checked in 0,00
sec, 61 MB used)
mv>
```

Module Verification (MV)

Assertion fails (I):

- select assertion
- start the debugger
- analyze the counterexample



```
Proof Status: mixed   Validity: up to date

clk      '1'
d_i      '0'
te/d_i   1'b0
te/q_o   1'b0

4 items total,

-I- Checking assertion 'sva/inst_dff_property_suite/dff_wrong_assertion'
-R- Assertion 'sva/inst_dff_property_suite/dff_wrong_assertion' fails (checked in 0,00
sec, 61 MB used)
mv>
```

SystemVerilog Assertions (SVA)

Overview

What is SVA?

- SystemVerilog Assertions (SVA) is a subset of SystemVerilog.
- It is a property (assertion) language describing design behaviors.
- It is suitable to express temporal design behaviors.
- It can be inserted into the HDL code or formulated in a standalone file.

Why do we need SVA?

- Verification using HDL is difficult
  - A request is granted exactly in two clock cycles

```
always @(posedge clk) begin
       if (req == 1'b1) cnt <= 1;
       else if (cnt == 1)
              cnt <= cnt + 1;
       else if (cnt == 2) begin
              if (grant == 1'b1)
              $display("request granted");
              else
              $display("request not granted")
       end
end
```

```
assert property (@(posedge clk) req |-> ##2 grant)
```

Why do we need SVA?

- Improve bug detection
- Improve the quality of the verification code
- SVA is an IEEE standard, and supported by tools (simulative or formal) from different vendors
- Systematic verification methodology

Property vs Assertion

- In SVA
  - A property is a formal description of some behavior of your design
  - An assertion is a directive to a verification tool to prove the validity of a given property.
- Some people use both notions to refer to the formal description. In this lab we also do so if the context is clear.

Immediate Assertions vs. Concurrent Assertions

- An immediate assertion is a non-temporal expression executed in a procedural code.
- It behaves like a procedural if statement and is evaluated when the control flow reaches the assertion.

```
assert (boolean expression) [action block]
```

- mostly useful in simulation flow.
- In our lab we focus only on concurrent assertions.

Immediate Assertions vs. Concurrent Assertions

- A concurrent assertion is a temporal expression and usually controlled by a clock.
- It is evaluated at the occurrence of the clock tick.

```
assert property (@(posedge clk) req |-> ##2 grant)
```

Assertion Overview

`assert, assume, cover`

`properties`

`sequences`

`Boolean expressions`

Boolean Expressions

- comparators

```
==, !=, >, <, >=, <=
```

- operands
  - design variables, literal constants
  - function calls returning values
- Boolean operators

```
&&, ||, !
```

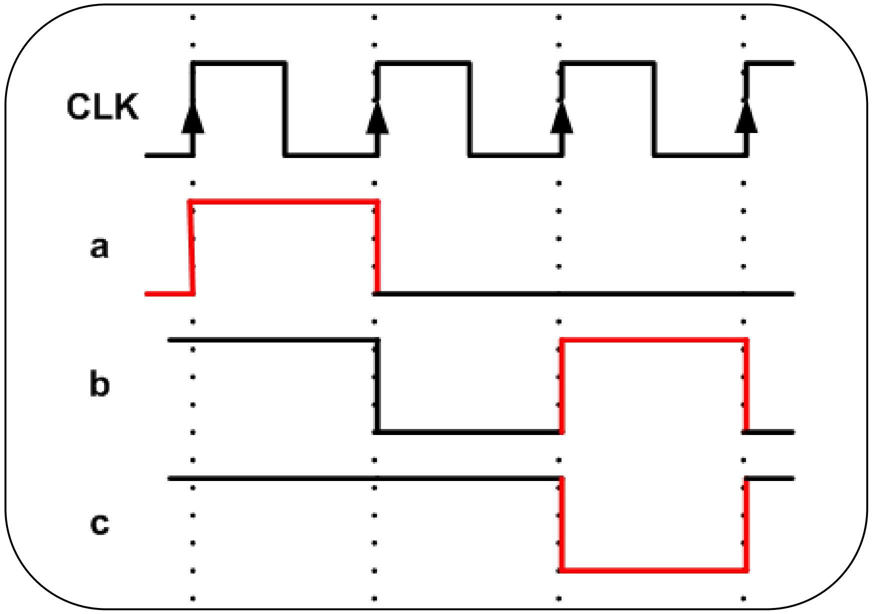
- Boolean constants

```
NON-zero value, 0
```

Sequences

- support formulating sequential behavior
- usually consist of Boolean expressions separated by cycle delays (##)

```
a ##2 b ##0 !c
```



Sequences (cycle delay range)

- ##[m:n], where m and n are constants and n>m

```
a ##[0:2] b ##0 !c
```

⟺

```
        either
a ##0 b ##0 !c
          or
a ##1 b ##0 !c
          or
a ##2 b ##0 !c
```

Sequences (repetition)

- consecutive repetition[*]

```
a ##1 b[*2] ##0 !c
```

⟺

```
a ##1 b ##1 b ##0 !c
```

- specify repetition range[*m:n]

```
a ##1 b[*0:2] ##0 !c
```

⟺

```
either
a ##1 `true ##0 !c
or
a ##1 b ##0 !c
or
a ##1 b ##1 b ##0 !c
```

Note: `true is not a keyword, but you may define it in Verilog syntax like this:
`define true 1

Named sequences

```
Basic syntax:
      sequence identifier[formal arguments];
            [variable declaration]
            sequence expressions;
      endsequence
```

```
Example:
      sequence myseq;
            a ##1 b[*2] ##1 c;
      endsequence
```

Sequence operators

- AND operation

```
s1 and s2;
// s1 and s2 must match
```

- OR operation

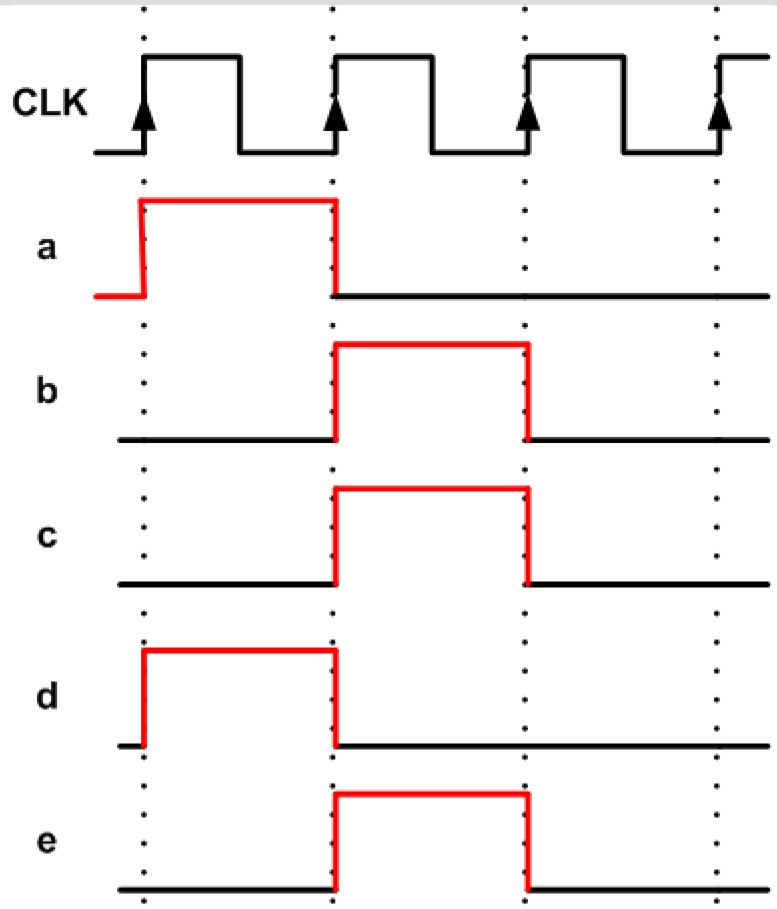
```
s1 or s2;
// s1 matches or s2 matches
```

- Note: s1 and s2 start at the same time
- NOT operation

```
not s1;
// inverts s1
```

Sequence operators (example)

```
sequence s1;
 a ##1 b ##0 c;
endsequence

sequence s2;
 d ##1 e;
endsequence

sequence s3;
 s1 and s2;
endsequence
```



Quiz: what ist the difference between the sequences "s1 and s2" and "s1 ##0 s2"?

Properties

```
Basic syntax:
              property identifier[formal arguments]
                      [local variables;]
                     sequences|property expressions;
              endproperty
```

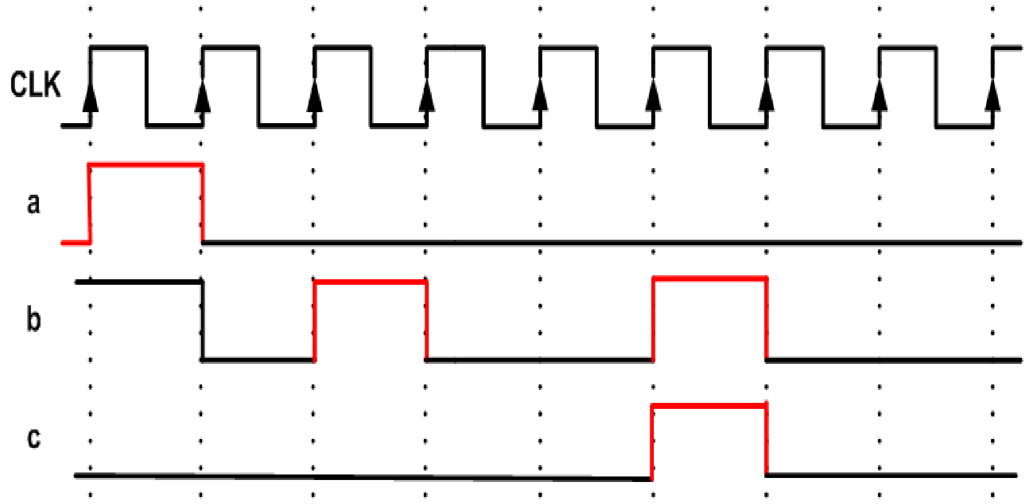
```
Example with implicator:

              property req_granted;
                req |=> ##5 grant;
              endproperty
```

Properties (implicator)

Overlapped implicator

|->

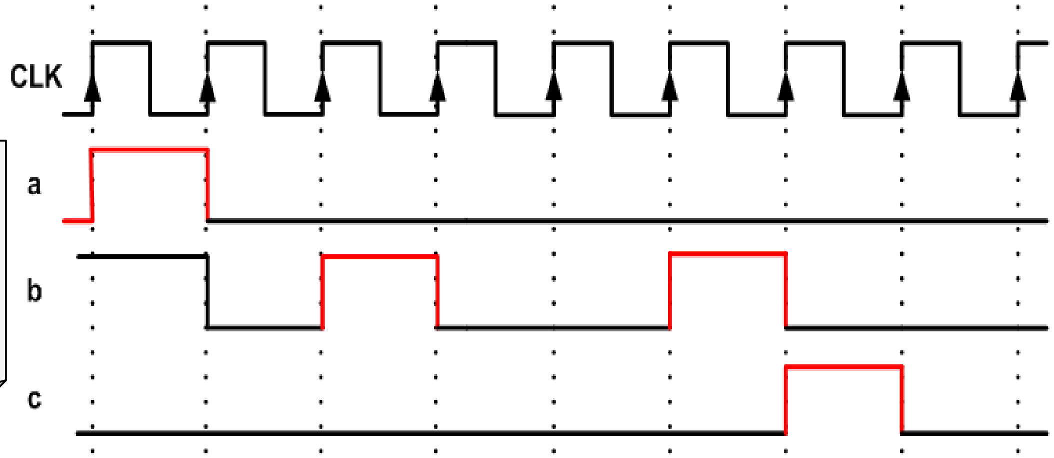
```
a ##2 b ##[1:3] b |-> c
```



NON-overlapped implicator

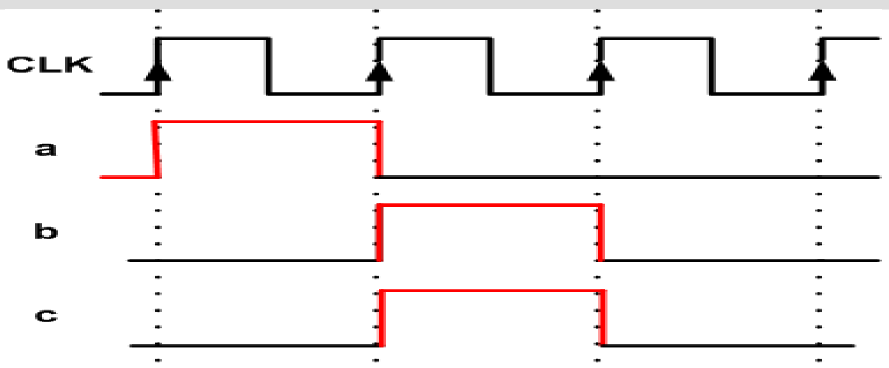
|=>

```
a ##2 b ##[1:3] b |=> c
```

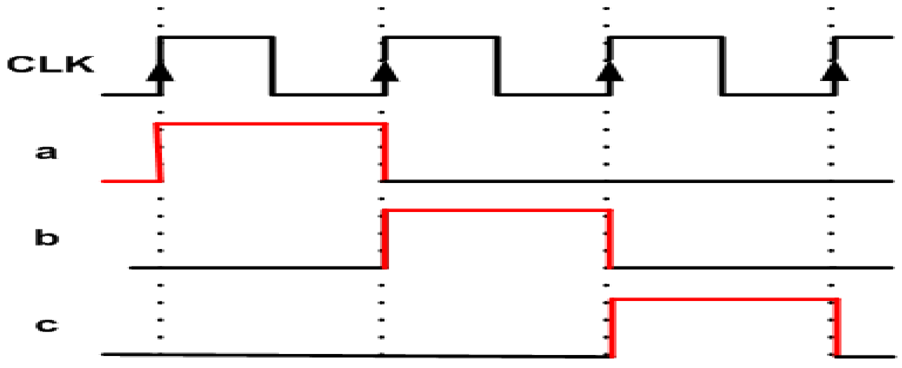


Quiz

```
a ##1 b |-> c
```



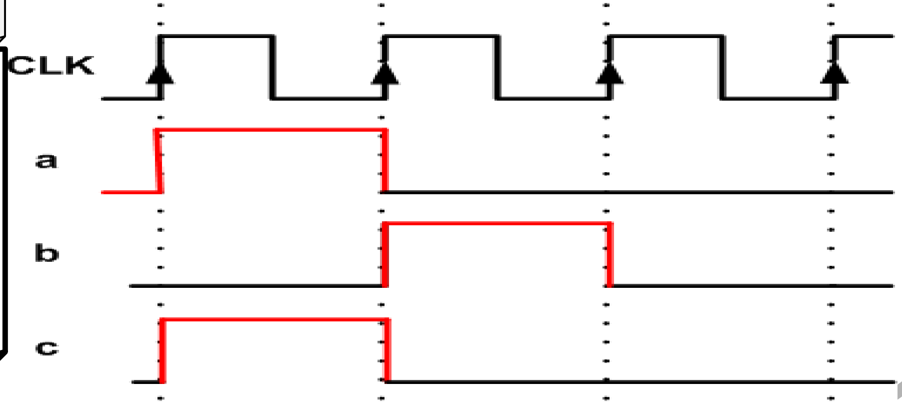
```
a ##1 b |=> c
```



```
not(a ##1 b) or c
```

In SystemVerilog Standard 2009, this formula can be expressed using a new keyword "implies", namely

```
a ##1 b implies c
```



System functions (e.g., $past)

- refer to the value of a signal in the past

```
property dff;
    q_o == $past(d);
           //same as $past(d,1)
endproperty
```

- other system functions: $onehot, $onehot0, $isunknown, $rose, $fell, $stable

Properties (define local variables)

```
property data_transfer;
      logic data_tmp;
      (valid_i, data_tmp = data_i) |=>
      ##2 (data_o == data_tmp);
endproperty
```

- The value of `data_i` is “frozen”, i.e., stored in the temporary variable `data_tmp` when `valid_i` is active.

Attention: Note the difference between assignment (=) and comparator (==)

ASSERT statement

- `assert` is a directive to the verification tool instructing it to verify that a given property is valid at all times.

```
label: assert property (@(posedge clk) myproperty);
```

- SVA provides a mechanism to disable an assertion during active reset (disable iff)

```
inst1: assert property (@(posedge clk)
       disable iff (!reset_n) myproperty);
```

ASSUME statement / Environment constraints

- **`assume`** is a directive to the verification tool instructing it to <u>assume</u> that a given property is valid at all times.

```
label: assume property (@(posedge clk) myproperty);
```

- All concurrent assertions are verified only for the scenarios (i.e., the input sequences to the design) for which the assumed properties hold.
- This allows formulating environment constraints.

Property module and BIND statement

- encapsulate your properties in one module as verification IP

```
module myip(a,b,c);
input logic a,c;
input logic[2:0] b;
// sequences
// properties
// assert directive
endmodule
```

- bind the verification IP to your RTL design

```
bind mydesign myip inst_my_ip(.*);

// explicit port mapping: by name (.a(HW_a),.b(HW_b))
// (.*) can only be used if the interfaces of your
// verification IP have the same names as the signals
// in the design.
```

Example: Verifying a FIFO

- The following two requirements need to be fullfilled by a synchronous FIFO:
  - *The full and empty flags cannot be active at the same time.*
  - *If there is no write operation on the FIFO then the content of FIFO is not altered.*

Example (cont.)

```
module fifo_property(clk,reset,full,empty,
         wr_valid,mem);

input logic clk;
input logic reset;
input logic full,empty;
input logic wr_valid;
              //indicates a write action
input logic [7:0] mem;

property requirement_1;
!(full == 1'b1 && empty == 1'b1);
endproperty
// continued on next slide
```

Example (cont.)

```
// continued from previous slide

property requirement_2;
wr_valid == 1'b0 |=> $stable(mem);
endproperty

inst1:assert property(@(posedge clk)
   disable iff (reset) requirement_1);
inst2:assert property(@(posedge clk)
   disable iff (reset) requirement_2);

endmodule
bind fifo fifo_property fifo_property_inst(.*);
```

Further advanced syntax

- Eduard Cerny, Surrendra Dudani, John Havlicek, Dmitry Korchemny :

  The Power of Assertions in SystemVerilog

  Springer, 2010

- Srikanth Vijayaraghavan, Meyyappan Ramanathan:

  A Practical Guide for SystemVerilog Assertions

  Springer, 2005

- Search the web for “systemverilog assertions tutorial”