Computer-Aided Design of Digital Systems (IAS0540)

WEEK 6

René Pihlak

Department of Computer Systems
School of Information Technology
Tallinn University of Technology

rene.pihlak@taltech.ee

2020-10-07

Introduction

Bugs

Validate, Verify, Test

The Reason for Verification

The Reason for Verification

'If we always designed correct systems, however, we would have no use for [...] verification (though we might still need formal specification). Most of the time, when we apply [...] "verification", we are applying it to a system that is actually incorrect — it has "bugs". [1]

Introduction

- Bugs
- Validate, Verify, Test

Quality Management

Quality management:

- validation
- verification
- testing

Validation, Verification, Testing

Validation, Verification, Testing

- validation

Validation, Verification, Testing

- **validation**
  - The assurance that a product, service, or system meets the needs of the customer

Validation, Verification, Testing

- **validation**
  - The assurance that a product, service, or system meets the needs of the customer
- **verification**

Validation, Verification, Testing

- **validation**
  - The assurance that a product, service, or system meets the needs of the customer
- **verification**
  - The evaluation of whether or not a product, service, or system complies with a regulation, requirement, **specification**, or imposed condition

Validation, Verification, Testing

- **validation**
  - The assurance that a product, service, or system meets the needs of the customer
- **verification**
  - The evaluation of whether or not a product, service, or system complies with a regulation, requirement, **specification**, or imposed condition
- **testing**

Validation, Verification, Testing

- **validation**
  - The assurance that a product, service, or system meets the needs of the customer
- **verification**
  - The evaluation of whether or not a product, service, or system complies with a regulation, requirement, **specification**, or imposed condition
- **testing**
  - The purpose of manufacturing tests is to assure that the product hardware contains no manufacturing defects that could adversely affect the product's correct functioning

Example of Quality Management

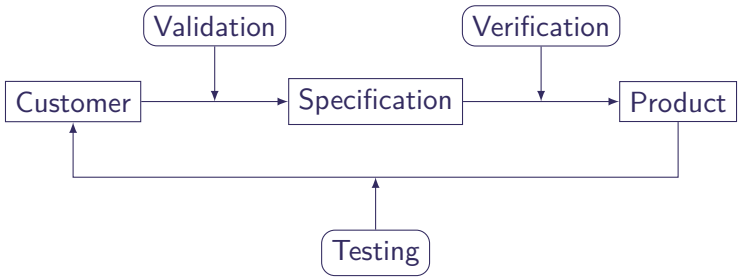
Example of Quality Management

Example of Quality Management

Example of Quality Management

Example of Quality Management

Example of Quality Management



Verification

Taxonomy of Verification

Verification Techniques

Hardware Code Coverage

Property Specification Language

Taxonomy of Verification

Classifications of Verification [2]

	VC Verification	Integration Verification
Intent Verification	✓	✓
Equivalence Verification	✓	✓

Classifications of Verification

- **Intent Verification**
  - verify that the designer's intended functionality has been correctly captured in the design
- **Equivalence Verification**
  - verify that the functionality of the various design levels created through the design process matches the functionality of the "golden model"
- **VC Verification**
  - verify the functionality of a virtual component, i.e., perform unit test
- **Integration Verification**
  - verify a system-on-chip (SoC) design that contains one or more VCs, i.e., perform system-level test of the SoC

Verification

- Taxonomy of Verification
- Verification Techniques
- Hardware Code Coverage
- Property Specification Language

Some Verification Techniques

Some Verification Techniques

- Deterministic Simulation

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation
- Physical Prototyping

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation
- Physical Prototyping
- Property/Model Checking

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation
- Physical Prototyping
- Property/Model Checking
- Coverage

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation
- Physical Prototyping
- Property/Model Checking
- Coverage
  - Functional Coverage

Some Verification Techniques

- Deterministic Simulation
- Random Pattern Simulation
- Static Functional Verification
- Emulation
- Physical Prototyping
- Property/Model Checking
- Coverage
  - Functional Coverage
  - Hardware Code Coverage

Verification

- Taxonomy of Verification
- Verification Techniques
- Hardware Code Coverage
- Property Specification Language

Types of Coverage

- **Statement coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**: how well a Boolean expression in an "if" condition or assignment has been exercised

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**: how well a Boolean expression in an "if" condition or assignment has been exercised
- **Path coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**: how well a Boolean expression in an "if" condition or assignment has been exercised
- **Path coverage**: which routes through sequential "if...else" and "case" constructs have been exercised

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**: how well a Boolean expression in an "if" condition or assignment has been exercised
- **Path coverage**: which routes through sequential "if...else" and "case" constructs have been exercised
- **Signal coverage**:

Types of Coverage

- **Statement coverage**: how many times each statement was executed
- **Toggle coverage**: which bits of the signals in the design have toggled
- **FSM coverage**: how many transitions of the Finite State Machine were processed
- **Visited state coverage**: how many states of the FSM were entered during simulation
- **Triggering coverage**: whether each process has been uniquely triggered by each of the signals in its sensitivity list
- **Branch coverage**: which "case" or "if...else" branches were executed
- **Expression coverage**: how well a Boolean expression in an "if" condition or assignment has been exercised
- **Path coverage**: which routes through sequential "if...else" and "case" constructs have been exercised
- **Signal coverage**: how well state signals or ROM addresses have been exercised

Verification

Property language standards

Property language standards are based on a branch of logic known as **temporal logic**.

Property language standards

Property language standards are based on a branch of logic known as **temporal logic**.

The advantage of using temporal logic to specify properties of reactive systems is that it enables us to reason about these systems in a simple way.

Property language standards

Property language standards are based on a branch of logic known as **temporal logic**.

The advantage of using temporal logic to specify properties of reactive systems is that it enables us to reason about these systems in a simple way.

That is, temporal logic eliminates the need to explicitly describe time when specifying relationships between system events.[3]

PSL: examples 1

$$\forall t.!(\texttt{grant1}(t) \quad \& \quad \texttt{grant2}(t))$$

which states that for all values of time t, `grant1` and `grant2` are mutually exclusive.

We write the property in a temporal language such as Property Specification Language — *PSL* — (which implicitly describes time) as follows:

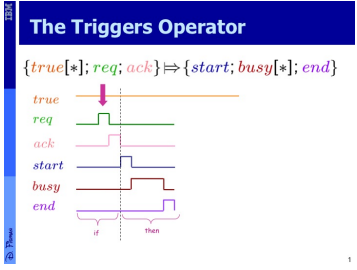
```
always!(grant1 & grant2)
```

which states that `grant1` and `grant2` never hold or evaluate to true at the same time.

PSL: examples 2

```
always (request -> eventually! grant)
```

```
{true[*]; req; ack}  |=> {start; busy[*]; end}
```



[4]

Practical Example

- Specification
- Design & Testbench
- Run Simulation
- Optional: Advanced Verification

eID SoC: Specification

Goal:
The eID SoC signals if previously unsigned document was successfully signed or not.

Design has three FSM states:

- `ST_LOGGED_OUT`: logged out
- `ST_LOGGED_IN`: logged in
- `ST_DOC_READ`: document is read

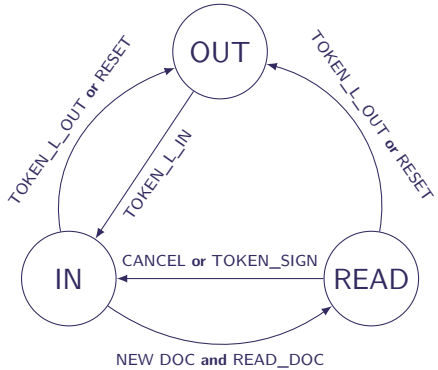
Output:

- `signature`: was signing successful? False on reset or cancel or logout.

Inputs:

- `reset`: user or system demands a reset
- `token_login`: user logged in successfully
- `token_logout`: user or system demands a logout
- `document`: there is a document
- `read_doc`: accept to read the document
- `token_sign`: user signed the document successfully
- `cancel`: user or system demands signing to be cancelled

Graph: FSM of eID SoC



Practical Example

- Specification
- Design & Testbench
- Run Simulation
- Optional: Advanced Verification

Design: eID (eid.vhd)

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use work.all;

entity EID is
port (
    clk:            in  std_logic;   -- CLOCK
    rst:            in  std_logic;   -- RESET
    token_login:    in  std_logic;   -- 1: log in success
    token_logout:   in  std_logic;   -- 1: log out success
    document:       in  std_logic;   -- 0: no doc to sign; 1: doc to sign
    read_doc:       in  std_logic;   -- 0: ignore doc; 1: read doc
    token_sign:     in  std_logic;   -- user input (password)
    cancel:         in  std_logic;   -- 0: not cancelled; 1: cancelled
    signature:      out std_logic    -- 0: not signed; 1: signed
);
end EID;

architecture behavior of EID is
-- FSM states
type FSM_TYPE is (ST_LOGGED_OUT, ST_LOGGED_IN, ST_DOC_READ);
signal STATE: FSM_TYPE := ST_LOGGED_OUT;
signal new_doc: std_logic := '0';
begin

FSM: process (rst, clk, document)
begin
    if (rst = '1') then
        signature <= '0';
        new_doc   <= '1';
        STATE     <= ST_LOGGED_OUT;
    elsif (cancel = '1') then
        signature <= '0';
        new_doc   <= '1';
        STATE     <= ST_LOGGED_OUT;
    elsif (document'event and document = '1') then
        new_doc   <= '1';
    elsif (clk'event and clk = '1') then
        case STATE is
            when ST_LOGGED_OUT =>
                if (token_login = '1') then -- login success
                    STATE <= ST_LOGGED_IN;
                end if;
            when ST_LOGGED_IN =>
                if (new_doc = '1' and read_doc = '1') then -- document needs signature
                    STATE       <= ST_DOC_READ;
                elsif (token_logout = '1') then -- logout
                    signature   <= '0';
                    new_doc     <= '1';
                    STATE       <= ST_LOGGED_OUT;
                end if;
            when ST_DOC_READ =>
                if (STATE = ST_LOGGED_IN) then -- require signature
                    STATE       <= ST_DOC_READ;
                elsif (token_sign = '1' and new_doc = '1') then
                -- sign the document
                    signature   <= '1';
                    new_doc     <= '0';
                end if;
        end case;
    end if;
end process;
end behavior;
```

Testbench: eID (tb_eid.vhd)

```
100 use std.textio.all;
101 library ieee;
102 use ieee.std_logic_1164.all;
103 use ieee.std_logic_textio.all;
104
105 entity E is
106 end E;
107
108 architecture A of E is
109 signal clk:              std_logic;   -- CLOCK
110 signal rst:              std_logic;   -- RESET
111 signal token_login:      std_logic;   -- 1: log in success
112 signal token_logout:     std_logic;   -- 1: log out success
113 signal document:         std_logic;   -- 0: no doc to sign; 1: doc to sign
114 signal read_doc:         std_logic;   -- 0: ignore doc; 1: read doc
115 signal token_sign:       std_logic;   -- user input (password)
116 signal cancel:           std_logic;   -- 0: not cancelled; 1: cancelled
117 signal signature:        std_logic;   -- 0: not signed; 1: signed
118
119 component EID
120 port(
121     clk:            in  std_logic;   -- CLOCK
122     rst:            in  std_logic;   -- RESET
123     token_login:    in  std_logic;   -- 1: log in success
124     token_logout:   in  std_logic;   -- 1: log out success
125     document:       in  std_logic;   -- 0: no doc to sign; 1: doc to sign
126     read_doc:       in  std_logic;   -- 0: ignore doc; 1: read doc
127     token_sign:     in  std_logic;   -- user input (password)
128     cancel:         in  std_logic;   -- 0: not cancelled; 1: cancelled
129     signature:      out std_logic    -- 0: not signed; 1: signed
130 );
131 end component;
132
133 begin
134     UUT: EID
135         port map (clk, rst, token_login, token_logout, document, read_doc, token_sign, cancel,
        signature);
136     TB: block
```

```
137     begin
138         process
139             file TV: TEXT is in "eid.tv";
140             variable L: line;
141             variable I_rst:              std_logic;
142             variable I_token_login:      std_logic;
143             variable I_token_logout:     std_logic;
144             variable I_document:         std_logic;
145             variable I_read_doc:         std_logic;
146             variable I_token_sign:       std_logic;
147             variable I_cancel:           std_logic;
148
149             begin
150                 readline(TV, L); -- read a line from 'eid.tv'
151                 read(L, I_rst);
152                 read(L, I_token_login);
153                 read(L, I_token_logout);
154                 read(L, I_document);
155                 read(L, I_read_doc);
156                 read(L, I_token_sign);
157                 read(L, I_cancel);
158                 if endfile(TV) then wait; -- THE END of TV
159                 end if;
160
161                 rst             <= I_rst;
162                 token_login     <= I_token_login;
163                 token_logout    <= I_token_logout;
164                 document        <= I_document;
165                 read_doc        <= I_read_doc;
166                 token_sign      <= I_token_sign;
167                 cancel          <= I_cancel;
168
169                 clk <= '0';
170                 wait for 10 ns;
171                 clk <= '1';
172                 wait for 10 ns;
173             end process;
174     end block;
175 end A;
```

Test Vectors: Initial Inputs (eid.tv)

```
100 0000000  # RESET, LOGIN, LOGOUT, DOCUMENT, READ_DOC, SIGN, CANCEL
101 1000000  # do reset
102 0100000  # do login
103 0001000  # new doc
104 0000100  # read doc
105 0000010  # sign
106 0001000
107 0000001
108 0100000
109 0001000
110 0010000
111 0000000
```

Practical Example

- Specification
- Design & Testbench
- Run Simulation
- Optional: Advanced Verification

Run Simulation/Verification (run_sim.do)

```
100 ##############################
101 #                            #
102 # Copyright (C) René Pihlak  #
103 #                            #
104 ##############################
105
106 if [file exists work] {
107     vdel -lib ./work -all
108 }
109
110 # make 'work' directory for the project if it does not exist
111 vlib work
112
113 # include and compile design/testbench files
114 vcom -2008 -coveropt 3 +cover +acc eid.vhd
115 # vcom -2008 -coveropt 3 +cover +acc eid.vhd -pslfile eid.psl
116 vcom -2008 -coveropt 3 +cover +acc tb_eid.vhd
117
118 # start simulation
119 vsim -coverage -assertdebug work.E
120
121 add wave -r sim:/e/UUT/*
122
123 WaveRestoreZoom {0 ps} {275 ns}
124
125 # restart sim
126 restart
127
128 # run sim for 275 ns
129 run 275 ns
```

Homework (Not Graded)

- Does the design capture the specifications correctly?
- Add additional test vectors to prove your verdict.
- If there are errors in the design, where are the errors? How can it be fixed?
- (Optional) Add PSL assertions to perform in depth analysis.

Practical Example

- Specification
- Design & Testbench
- Run Simulation
- Optional: Advanced Verification

Advanced Verification: PSL (eid.psl)

```
vunit check_eid(eid(behavior))
{
    -- set CLOCK
    default clock is rising_edge(clk);

    -- properties
    --
    -- RESET
    property rst_to_logout1 is always ({rst} |-> {STATE = ST_LOGGED_OUT});
    property rst_to_logout2 is always (rst -> next(STATE = ST_LOGGED_OUT));
    property rst_to_logout3 is always (rst -> next!(STATE = ST_LOGGED_IN)); -- incorrect syntax
    property rst_to_logout3a is always (rst -> next(STATE /= ST_LOGGED_IN));
    property rst_to_logout3b is never ({rst; STATE = ST_LOGGED_IN});
    property rst_to_logout4 is always ({STATE = ST_LOGGED_IN: rst} |=> {STATE = ST_LOGGED_OUT});

    -- SIGN
    property get_signature is always (token_sign -> eventually! (signature));

    -- assert properties
    a_rst_logout1  : assert rst_to_logout1 report "rst -> logout 1";
    a_rst_logout2  : assert rst_to_logout2 report "rst -> logout 2";
    a_rst_logout3  : assert rst_to_logout3 report "rst -> logout 3";
    a_rst_logout3a : assert rst_to_logout3a;
    a_rst_logout3b : assert rst_to_logout3b;
    a_rst_logout4  : assert rst_to_logout4 report "rst -> logout 4";
    a_get_sign     : assert get_signature report "document is signed?";

    -- sequences
    sequence seq_1 is {STATE=ST_LOGGED_OUT; STATE=ST_LOGGED_IN[*]; STATE=ST_DOC_READ[*]; STATE=ST_LOGGED_IN};

    -- coverage
    s_1:   cover {STATE=ST_LOGGED_OUT} report "missing state?";
    s_2:   cover {STATE=ST_LOGGED_IN}  report "missing state?";
    s_3:   cover {STATE=ST_DOC_READ}  report "missing state?";
    cov_1: cover seq_1;
}
```

References

- K. McMillan, Verification of Digital and Hybrid Systems, 1st ed., ser. NATO ASI Series 170. Springer-Verlag Berlin Heidelberg, 2000, ch. 1.
- B. Bailey, G. Martin, and T. Anderson, Taxonomies for the development and verification of digital systems, 1st ed. Springer, 2005.
- D. Perry and H. Foster, Applied Formal Verification: For Digital Circuit Design, 1st ed. McGraw-Hill Professional, 2005.
- Fismand, "Property specification language," https://en.wikipedia.org/wiki/Property_Specification_Language, 2004.