Processor Design

José María Arnau
Validation vs Verification

- **Validation**
  - Ensure that system meets operational needs of the user
  - Are we building the right system?

- **Verification**
  - Check that system conforms to specification
  - Are we building the system right?
Hardware Verification

- Verification is a process used to demonstrate the functional correctness of a design
- 60%-80% of effort in hardware design is dedicated to verification
- 80% of all written code is in the verification environment
- Most hardware engineers spend most of their time doing verification
- Verification is on the critical path!
Verification & Debug Consume Majority of Time

Mike Stellfox. Verification is a Problem, but is Debug the Root Cause?
Verification Time is on the Rise

At this rate... In 25 years, ALL of a designer’s time will be devoted to verification.

Time Design Engineers Spends Doing:
- Design
- Verification

Majority of Flaws are Functional

Harry Foster. Verification is a Problem, but is Debug the Root Cause?
Late-Stage Bugs Are More Costly

**Effort to Debug & Fix**

- **Post-Silicon Validation Engineers** find/debug post-silicon bugs
  - Silicon or silicon prototype, leverage SLV environment

- **Verification Engineers** find/debug module & chip bugs (CDV/MDV)
  - Simulation spec-based Coverage or Metric Driven Verification

- **System Validation Engineers** find/debug system bugs (SLV)
  - Acceleration/Emulation for system environment, HW/SW

**Time to bug discovery**

Mike Stellfox. Verification is a Problem, but is Debug the Root Cause?
The Relative Cost of Finding Bugs

Cost To Fix

$10,000,000

$1,000,000

$100,000

$10,000

$1,000

$100

$10

$1

Design Cycle

Initial Design
Design Review
Layout
Tape Release
Early Silicon
Sampling
Volume Production

Silicon Debug, Doug Josephson and Bob Gottlieb, (Paul Ryan)
Achieving First Silicon Success

Harry Foster. Verification is a Problem, but is Debug the Root Cause?
Verification Efficiency vs Effectiveness

http://www.brianbailey.us/blog/verification-101-verification-about/
Hardware Verification

- **Functional** Verification
  - Task of verifying that the RTL conforms to specification
  - Based on RTL simulation

- **Formal** Verification
  - Proving or disproving the correctness of the system with respect to a specification
RTL Simulators

- Commercial simulators
  - Synopsys VCS
  - Mentor Graphic’s ModelSim
- Open source
  - Icarus Verilog
  - GHDL
  - Verilator
What is a testbench?

- A “testbench” is the code used to create a determined input sequence to a design and then observe the response
  - Written in VHDL, Verilog, C++, Python...
- Verification challenge:
  - Find what input patterns to supply to Design Under Test and what is expected for the output of a properly working design
Directed Tests

- Pre-defined stimulus to test particular functionality
- Testbench implements specs to check outputs
- Very labour intensive
- Hard to model all cases

```
User defined test-vectors

Driver

Design Under Test (DUT)

Checker

User defined expected output
```
Directed Tests

- Pre-defined stimulus to test particular functionality
- Testbench implements specs to check outputs
- Very labour intensive
- Hard to model all cases

Only find bugs you are looking for!
Constrained Random Verification

- Stimulus created automatically using constraints
- Write constraints to generate random stimuli
  - e.g. “Valid range of address is XX to YY”
  - e.g. “Valid data is any 8 bit data with a parity bit. 90% of time I want parity set correctly, 10% incorrectly”
- Run test with multiple seeds for larger coverage
- Able to cover large portion of state space quickly
  - Corner cases may still be difficult to find
What to Randomize

- Input data
- Device configuration
- Delays
- Transaction length
Constrained Random Verification

Constrained Random Generator

Driver

Design Under Test (DUT)

Reference model (emulator)

Checker

Input stimuli

Expected output

DUT output
Constrained Random Verification

Constrained Random Generator

Driver

Reference model (emulator)

Checker

Design Under Test (DUT)

Which functionality has been tested?
Coverage

- Measures which design features have been exercised
- Code coverage metrics:
  - e.g. line, path, toggle, FSM
- Functional coverage
  - e.g. Ensuring that all possible interactions with DUT have been tested
Directed vs Random Test

Coverage vs Time

100%

Time

Directed Test

Random Test

Chris Spear. SYSTEMVERILOG FOR VERIFICATION
A Guide to Learning the Testbench Language Features
Assertions

• Assertions specify designer’s intent in a formal property
• Assertion based verification (ABV) uses assertions with both dynamic simulation or static formal verification
• Example:
  - always @(*) assert(!overflow);
Formal Verification

- Can “prove” correctness about design blocks
- Exhaustively check state-space of a design for violations of properties (assertions)
  - E.g. “error signal should never be high”
- Does not execute design, proves property mathematically
- Good at finding corner case behaviors
- Requires high-effort, specialized knowledge
- May be impractical for large designs
SystemVerilog

- HDL & Verification language based on Verilog
  - First appeared in 2002
  - Subsumes old Verilog standard
- “New” verification features:
  - Constrained random stimulus generation
  - Assertions
  - Coverage
  - Object-oriented programming
  - Convenient interface to foreign languages (C/C++)
Open Verification Library

- Library of assertions across many HDL languages
  - Verilog, VHDL, SystemVerilog...
- Standardized, supported by major simulators
- Support common assertion structures using module/parameter instantiations

```vlog
ovl_never #(`OVL_ERROR, `OVL_ASSERT, "Register A < Register B", `OVL_COVER_ALL)
  valid_checker_inst(clk, reset_n, regA < regB);
```
Universal Verification Methodology

- Standardized framework for coverage-driven functional verification
  - Methodology employs test generation, self-checking testbenches, coverage metrics...
  - Promotes re-use
  - Supported by all major EDA vendors

- Framework provides:
  - Object-oriented base classes in SystemVerilog for transactions, drivers, monitors, etc.
Cocotb

- Library for digital logic verification in Python
  - Coroutine cosimulation testbench
  - Python interface to RTL simulators
  - Human-friendly alternative to Verilog/VHDL
- Verification testbenches are software!
- Benefits of Python
  - Extremely simple and easy to learn, but very powerful
  - Large standard library and huge ecosystem
  - Easy to find Python developers

https://github.com/cocotb/cocotb
Cocotb – Basic Architecture

- DUT runs in standard simulator
- Cocotb provides interface between simulator and Python
- Python testbench code can:
  - Reach into DUT hierarchy and change values
  - Wait for simulation time to pass
  - Wait for a rising or falling edge of a signal
Cocotb - Example

Verilog code (add.v)

```verilog
module add(
    input wire [7:0] a,
    input wire [7:0] b,
    output wire [8:0] c);

assign c = a + b;
endmodule
```
Cocotb - Example

Verilog code (add.v)

```verilog
module add(input wire [7:0] a,
           input wire [7:0] b,
           output wire [8:0] c);
    assign c = a + b;
endmodule
```

Python code (add_test.py)

```python
import cocotb
from cocotb.triggers import Timer

@cocotb.test()
def add_test(dut):
    dut.a <= 3
    dut.b <= 5
    yield Timer(1, "ns")
    assert dut.c == 8
```
Cocotb - Example

**Verilog code (add.v)**

```verilog
module add(input wire [7:0] a,
           input wire [7:0] b,
           output wire [8:0] c);
    assign c = a + b;
endmodule
```

**Python code (add_test.py)**

```python
import cocotb
from cocotb.triggers import Timer

@cocotb.test()
def add_test(dut):
    dut.a <= 3
    dut.b <= 5
    yield Timer(1, "ns")
    assert dut.c == 8
```

**Makefile**

```
VERILOG_SOURCES = add.v
TOPLEVEL=add
MODULE=add_test
include $(shell cocotb-config --makefiles)/Makefile.inc
include $(shell cocotb-config --makefiles)/Makefile.sim
```
Cocotb - Example

Running tests with Cocotb v1.2.0rc1 from /usr/local/lib/python2.7/dist-packages
Seeding Python random module with 1569503113
Found test add_test
Running test 1/13 add_test
Starting test: "add_test"
Description: None
Test Passed: add_test
Passed 1 tests (0 skipped)
******************************************************************************
** TEST PASS/FAIL SIM TIME(NS) REAL TIME(S) RATIO(NS/S) **
******************************************************************************
** add_test.add_test PASS 1.00 0.00 1017.32 **
******************************************************************************

******************************************************************************
** ERRORS : 0 **
******************************************************************************
** SIM TIME : 1.00 NS **
** REAL TIME : 0.00 S **
** SIM / REAL TIME : 533.89 NS/S **
******************************************************************************

Shutting down...
Cocotb - Triggers

• Cosimulation
  – DUT and testbench simulated independently
  – Communication through VPI/VHPI interfaces, represented by cocotb “triggers”
  – When Python code is executing, simulation time is not advancing
  – When a trigger is yielded, testbench waits until condition is satisfied before resuming execution

• Some available triggers:
  – Timer(time, unit)
  – RisingEdge(signal)
  – ClockCycles(signal, num)
Coroutines

- Cocotb uses a cooperative multitasking architecture
- Tests can call functions
  - If they want to consume simulation time, must be coroutines

```python
import cocotb
from cocotb.triggers import Timer

@cocotb.coroutine
def Driver(signal1, signal2, v1, v2):
    signal1 <= v1
    signal2 <= v2
    yield Timer(1, "ns")

@cocotb.coroutine
def Checker(signal, expected):
    yield Timer(1, "ns")
    assert signal == expected

@cocotb.test()
def add_test(dut):
    cocotb.fork(Driver(dut.a, dut.b, 3, 5))
    cocotb.fork(Checker(dut.c, 8))
    yield Timer(1, "ns")
```
Cocotb-coverage

• Adds support for modern verification techniques
  – Constrained Random Verification
  – Functional Coverage

https://github.com/mciepluc/cocotb-coverage
SV vs Cocotb - CRV

Marek Cieplucha and Witold Pleskacz. New Constrained Random and Metric-Driven Verification Methodology using Python
Cocotb - Coverage

- A tree structure
- Coverage primitive
  - Function decorator
  - User can define own coverage types
  - SystemVerilog originated:
    - CoverPoint
    - CoverCross
SV vs Cocotb - Coverage

Marek Cieplucha and Witold Pleskac. New Constrained Random and Metric-Driven Verification Methodology using Python
Cocotb-coverage - Example

```python
import cocotb
from cocotb.triggers import Timer
from cocotb_coverage import crv

class rand_input(crv.Randomized):
    def __init__(self):
        crv.Randomized.__init__(self)
        self.a = 0
        self.b = 0
        self.add_rand("a", list(range(256)))
        self.add_rand("b", list(range(256)))

@cocotb.coroutine
def Driver(signal1, signal2, N):
    ri = rand_input()
    for _ in range(N):
        ri.randomize()
        signal1 <= ri.a
        signal2 <= ri.b
        yield Timer(1, "ns")

@cocotb.coroutine
def Checker(dut, N):
    for _ in range(N):
        yield Timer(1, "ns")
    assert dut.c == (dut.a.value + dut.b.value)
```

```python
@cocotb.test()
def add_test(dut):
    N = 1000
    cocotb.fork(Driver(dut.a, dut.b, N))
    yield cocotb.fork(Checker(dut, N)).join()
```
Cocotb-coverage - Example

```python
import cocotb
from cocotb.triggers import Timer
from cocotb_coverage import crv

class rand_input(crv.Randomized):
    def __init__(self):
        crv.Randomized.__init__(self)
        self.a = 0
        self.b = 0
        self.add_rand("a", list(range(256)))
        self.add_rand("b", list(range(256)))

@cocotb.coroutine
def add_test(dut):
    N = 1000
    cocotb.fork(Driver(dut.a, dut.b, N))
    yield cocotb.fork(Checker(dut, N)).join()

@cocotb.coroutine
def Driver(signal1, signal2, N):
    ri = rand_input()
    for _ in range(N):
        ri.randomize()
        signal1 <= ri.a
        signal2 <= ri.b
        yield Timer(1, "ns")

@cocotb.coroutine
def Checker(dut, N):
    for _ in range(N):
        yield Timer(1, "ns")
    assert dut.c == (dut.a.value + dut.b.value)
```

Have we covered the overflow case?
import cocotb
from cocotb import logging
from cocotb.triggers import Timer
from cocotb_coverage import crv
from cocotb_coverage.cov_types import *

class rand_input(crv.Randomized):
    def __init__(self):
        crv.Randomized.__init__(self)
        self.a = 0
        self.b = 0
        self.add_rand("a", list(range(256)))
        self.add_rand("b", list(range(256)))

@CoverPoint("top.overflow",
            xf = lambda ri:(ri.a+ri.b)>255,
            bins = [0, 1])
def sample_coverage(ri):
    pass

@cocotb.coroutine
def Checker(dut, N):
    for _ in range(N):
        yield Timer(1, "ns")
        assert dut.c == (dut.a.value +
                         dut.b.value)

@cocotb.test()
def add_test(dut):
    N = 1000
    cocotb.fork(Driver(dut.a, dut.b, N))
    th = cocotb.fork(Checker(dut, N))
    yield th
    log = logging.getLogger("cocotb.test")
    coverage_db.report_coverage(log.info,
                                bins=True)
Cocotb-coverage - Example

Running tests with Cocotb v1.2.0 from /usr/local/lib/python3.6/dist-packages
Seeding Python random module with 1569512723
Found test add_test

Running test 1/1: add_test
Starting test: "add_test"
Description: None
top: <cocotb_coverage.coverage.CoverItem object at 0x7f3a494a47b8>, coverage=2, size=2
top.overflow: <cocotb_coverage.coverage.CoverPoint object at 0x7f3a494ab080>, coverage=2, size=2
  BIN 0 : 538
  BIN 1 : 462
Test Passed: add_test
Passed 1 tests (0 skipped)

** TEST PASS/FAIL SIM TIME(NS) REAL TIME(S) RATIO(NS/S) **
******************************************************************************
** add_test.add_test PASS 1000.00 0.44 2275.00 **
******************************************************************************

******************************************************************************
** ERRORS : 0 **
******************************************************************************
** SIM TIME : 1000.00 NS **
** REAL TIME : 0.50 S **
** SIM / REAL TIME : 1999.39 NS/S **
******************************************************************************

Shutting down...
Verilator

- Fastest free Verilog simulator
  - Suitable for large designs
  - Compiles Verilog to C++
  - Multithreaded since 4.0
- Support for synthesizable SystemVerilog
- Testbenches written in C++
- https://www.veripool.org/wiki/verilator
Verilator - Example

- Translate to C++ class
  - `verilator -cc Convert.v`

```plaintext
module Convert;
  input clk
  input [31:0] data;
  output [31:0] out;
  initial $display("Hello flip-flop");
  always_ff @(posedge clk)
    out <= data;
endmodule
```

```plaintext
#include "verilated.h"

class VConvert {
  bool clk;
  uint32_t data;
  uint32_t out;

  void eval();
  void final();
}
```
`timescale 1ns/1ps

`default_nettype none

module dotprod(input wire clk,
               input wire rst,
               input wire [7:0] Ai,
               input wire [7:0] Bi,
               output wire [25:0] out);

    reg [25:0] accum;

    always @(posedge clk)
    begin
      if (rst) accum <= 26'd0;
      else accum <= accum + Ai * Bi;
    end

    assign out = accum;

endmodule // dotprod
DotProd Testbench - Verilator

```c
#include "Vdotprod.h"
#include "verilated.h"

// Generates two random vectors A and B
struct RandomVectors {
    vector<unsigned char> A;
    vector<unsigned char> B;
    RandomVectors(int N) : A(N), B(N) {
        generate(A.begin(), A.end(), rand);
        generate(B.begin(), B.end(), rand);
    }
};

// Control coverage
struct SampleCoverage {
    vector<unsigned int> bins;
    SampleCoverage() : bins(1025, 0) {}
    void sample(int N) { bins[N]++;
    }
    void printCoverage(ostream& out) {
        int bins_hit = 0;
        for (size_t i = 0; i < bins.size(); i++)
            if (bins[i])
                bins_hit++;
        out << "Bins covered: " << bins_hit << " / " << 1024 << endl;
        out << "Coverage = " << bins_hit / 1024.0f * 100.0f << "%" << endl;
    }
};

// Advance one cycle of RTL simulation
void runCycle(Vdotprod *dut, int halfCycle=5) {
    dut->clk = 1;
    for (int i = 0; i < halfCycle; i++)
        dut->eval();
    dut->clk = 0;
    for (int i = 0; i < halfCycle; i++)
        dut->eval();
}
```
int main(int argc, char **argv) {
    Verilated::commandArgs(argc, argv);
    Vdotprod* dut = new Vdotprod;
    int NUM_EXECUTIONS = atol(argv[1]);

    SampleCoverage coverage;
    for (int i = 0; i < NUM_EXECUTIONS; i++) {
        int N = (rand() % 1024) + 1;
        int expected_out = 0;

        RandomVectors randVecs(N); // Generate random vectors
        dut->rst = 1;
        runCycle(dut); // Reset cycle
        dut->rst = 0;

        for (size_t j = 0; j < randVecs.A.size(); j++)
        {
            dut->Ai = randVecs.A[j];
            dut->Bi = randVecs.B[j];
            runCycle(dut);
        }

        cout << "Test " << i << ": dut.out" << dut->out << ", expected.out" << expected_out << endl;
        assert(dut->out == expected_out); // Check output

        coverage.sample(N); // Update coverage info
    }

    coverage.printCoverage(cout);
}
CRAVE: CRV in C++

- Constrained Random Verification Environment
- Constrained random stimuli generator
- SystemVerilog inspired syntax
- Hard/soft constraints
- Fast generation
- Available in github:
  - https://github.com/agra-uni-bremen/crave-bundle
CRAVE - Example

```cpp
class RandomVectors : public rand_obj
{
    vector< randv<unsigned int> > A;
    vector< randv<unsigned int> > B;
    RandomVectors(int N) : rand_obj(), A(N), B(N) {
        for (int i = 0; i < N; i++) {
            constraint(A[i]() < 256);
            constraint(B[i]() < 256);
        }
    }
};

class RandTransaction : public rand_obj
{
    randv<bool> we;
    randv<unsigned short> addr;
    randv<unsigned char> len;
    randv<unsigned long long> wdata;
    RandTransaction() : rand_obj() {
        constraint(addr() < (1<<14));
        constraint(len() < 4);
        constraint(if_then(len() == L_HALF), (addr() % 2) == 0);
        constraint(if_then(len() == L_WORD), (addr() % 4) == 0);
        constraint(if_then(len() == L_DWORD), (addr() % 8) == 0);
    }
};
```
CRAVE Architecture

C++ Constraint Specification

Intermediate Representation (Constrained Objects)

Pre-generation Analyses

Generation

Post-generation Analyses

Constraint Partitioner
Soft Constraint Analyzer
Distribution Solver
Solver Parallelizer
Constraint Debugger
Coverage Analyzer

Multi-solver backend

SMT Solvers
BDD Solver
CRAVE - Soft Constraints

- Hard constraints must always be satisfied
- Soft constraints shall be satisfied unless contradicted by
  - A hard constraint
  - A soft constraint with higher priority

```cpp
class packet : public rand_obj {
    randv<unsigned int> size;
    randv<unsigned int> dest_addr;
    packet() : rand_obj() {
        constraint(dest_addr() <= 0xFFFF0000);
        soft_constraint(size() >= 10);
        soft_constraint(size() < 1000);
    }
};

class short_packet : public packet {
    short_packet() : packet() {
        soft_constraint(size() >= 5);
        soft_constraint(size() < 10);
    }
};
```
CRAVE - Distribution Constraint

- Uniform distribution is not always wanted
  - E.g. $a < 10$ or $b < 10$ will be extremely rare
  - Hard to reach coverage closure
- Distribution constraints
  - User-defined biases to create interesting stimuli (corner cases)

```cpp
class my_rand_obj: public rand_obj {
  randv<unsigned int> a, b, c;
  my_rand_obj(): rand_obj() {
    constraint(if_then(a() < 10, c() == 0));
    constraint(if_then(b() < 10, c() == 1));
    constraint(a() <= 1000000000);
    constraint(b() <= 1000000000);
    constraint(c() <= 1000000000);
  }
}
class my_rand_obj_ext: public my_rand_obj {
  my_rand_obj_ext(): my_rand_obj() {
    constraint(dist(a()),
      distribution<unsigned int>::create
      (weighted_range<unsigned int>(0, 9, 30)) // 30%
      (weighted_range<unsigned int>(10, 1000000000, 70)) // 70%
    );
    constraint(dist(b()),
      distribution<unsigned int>::create
      (weighted_range<unsigned int>(0, 9, 50)) // 50%
      (weighted_range<unsigned int>(10, 1000000000, 50)) // 50%
    );
  }
};
```
Other solutions for CRV and FC

- **UVM-SystemC**
- SystemC Verification Library (**SVL**)  
- Functional Coverage for SystemC (**FC4SC**)
Other solutions for CRV and FC

- UVM-SystemC
- SystemC Verification Library (SVL)
- Functional Coverage for SystemC (FC4SC)

*HOW STANDARDS PROLIFERATE:*  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

**SITUATION:**  
There are 14 competing standards.

**14?! RIDICULOUS!**  
We need to develop one universal standard that covers everyone’s use cases. **YEAH!**

**SOON:**  
**SITUATION:**  
There are 15 competing standards.
Assertions

- Capture the knowledge about how a design should operate
- Increase the observability of a design
- Each assertion specifies both legal and illegal behavior of a circuit structure inside the design
- Assertions in English:
  - The result should not overflow
  - Signal “a” must be less than...
Assertion Based Verification

- Methodology to detect bugs quickly and close to source
- Increased observability
Traditional Verification

- Apply vectors to stimulate incorrect behavior
- Trace back through waveforms to locate the source

Mentor Graphics. “Assertion Based Verification”
Assertion Based Verification

- Observe bug when propagated to output
- Catch bugs at or near source of the problem

Mentor Graphics. “Assertion Based Verification”
SystemVerilog Assertions (SVA)

- Powerful language to describe assertions
- Two types of assertions
  - Immediate assertions
  - Concurrent assertions
- Perform test on an aspect of the design
  - Pass or fail statements can be executed
- Typically ignored during synthesis
- Partial SVA support in Verilator
SystemVerilog Assertions (SVA)

- Immediate assertions
  - Execute immediately, in zero simulation time
  - Can be placed anywhere procedural statements can be placed: within always blocks, initial blocks...

```systemverilog
assert (expression) [pass_statement;] [else fail_statement;]
```
SystemVerilog Assertions (SVA)

- Immediate assertions

```verilog
module example_1 (  
    input ifcond, a, b,  
    output logic if_out  
);

always_comb begin
    assert (^ifcond !== 1'bx);
    else $error("ifcond = X");
    if (ifcond)
        if_out = a;
    else
        if_out = b;
end
endmodule
```
SystemVerilog Assertions (SVA)

- Immediate assertions

```verilog
module example_1 (  
    input ifcond, a, b,  
    output logic if_out  
);

    always_comb begin  
        assert (^ifcond !== 1'bX);  
            else $error("ifcond = X");  
        if (ifcond)  
            if_out = a;  
        else  
            if_out = b;  
    end  
endmodule

module example_2 (  
    input a, b, c, d,  
    input [1:0] sel,  
    output logic out  
);

    always_comb begin  
        assert (^sel !== 1'bX);  
            else $error("case_Sel = X");  
        case (sel)  
            2'b00 : out = a;  
            2'b01 : out = b;  
            2'b10 : out = c;  
            2'b11 : out = d;  
        endcase  
    end  
endmodule
```

Don Mills and Stuart Sutherland. SystemVerilog Assertions Are For Design Engineers Too!
SystemVerilog Assertions (SVA)

- Concurrent assertions
  - Use a clock to trigger the assertion evaluation
  - Evaluate conditions over time

```verilog
assert property (property_expr) [pass_statement;] [else fail_statement;]
```
SystemVerilog Assertions (SVA)

- Concurrent assertions

```vhdl
example_3: assert property @(posedge clk) ( req ##1 grant ##10 !req ##1 !grant);
else $error("bus request failed");
```
SystemVerilog Assertions (SVA)

- Concurrent assertions

```verilog
eexample_3: assert property @(posedge clk) ( req ##1 grant ##10 !req ##1 !grant);
else $error("bus request failed");

property bus_req_prop2;
 @(posedge clk) req ##1 grant ##10 !req ##1 !grant;
endproperty

eexample_4: assert property (bus_req_prop2);
else $error("bus request failed");
```

Don Mills and Stuart Sutherland. SystemVerilog Assertions Are For Design Engineers Too!
When is Verification Done?

- Never truly done on complex designs
- Functional verification can only show presence of errors, not their absence
- Given enough time, errors will be uncovered
Bibliography

- Brian Keng. An Overview of Modern Functional Verification and Debug
- Ben Rosser. Cocotb: a Python-based digital logic verification framework
- Marek Cieplucha and Witold Pleskacz. New Constrained Random and Metric-Driven Verification Methodology using Python
- http://www.cse.psu.edu/~vxn9/verify/protected/slides/ncstate/Ch1.ppt
Bibliography

- CRAVE 2.0: The Next Generation Constrained Random Stimuli Generator for SystemC
- Mentor Graphics. “Assertion Based Verification”
- Don Mills and Stuart Sutherland. SystemVerilog Assertions Are For Design Engineers Too!