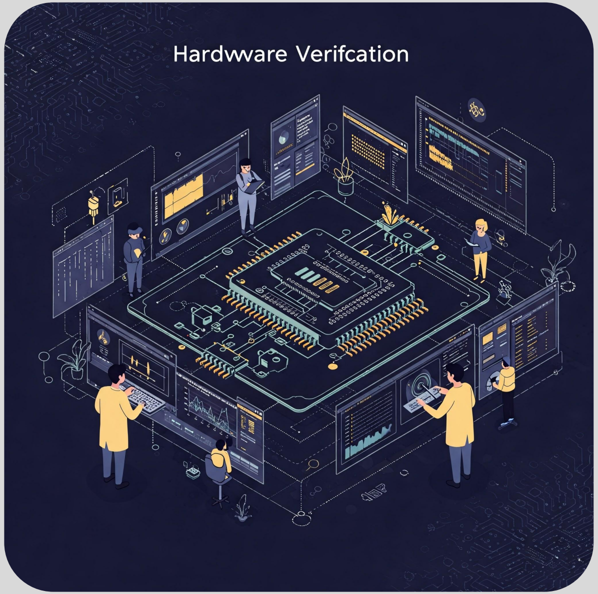
UVM: Verifying Digital Designs Effectively



- Hardware verification ensures correct IC functionality.
- UVM is the industry standard for verification.
- UVM promotes reusability and modularity.
- pyUVM implements UVM with Python and Cocotb.
- pyUVM leverages Python's ecosystem and syntax.

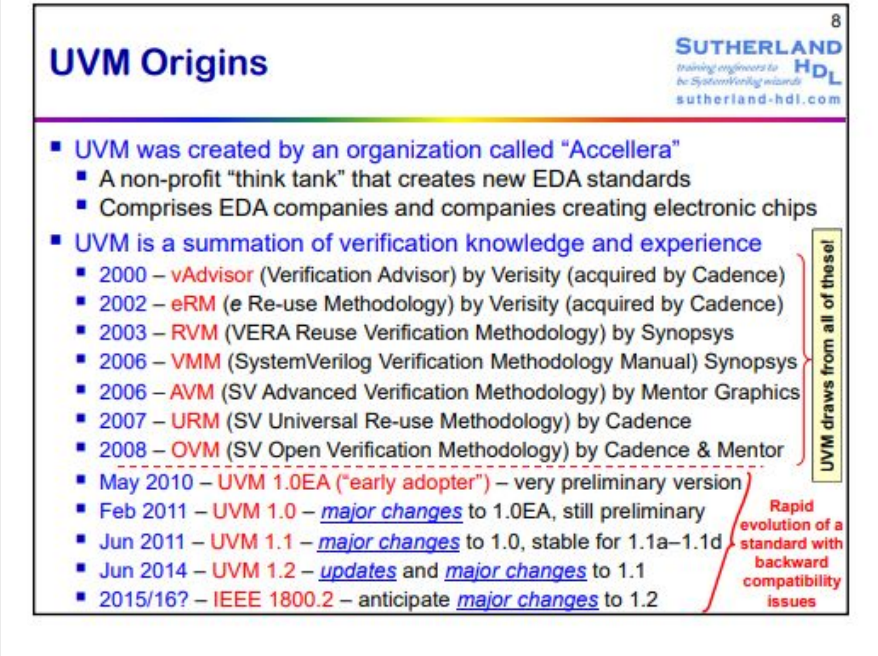
Agenda

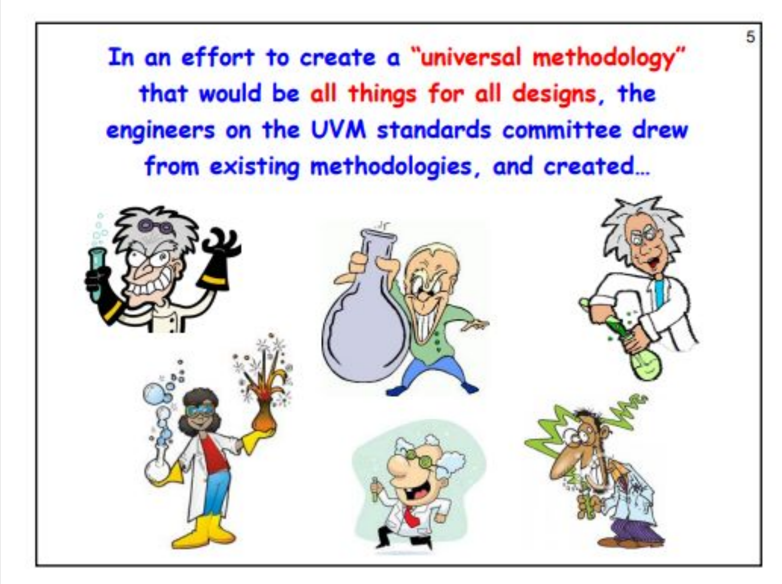
1. Introduction to Hardware Verification and UVM
2. History of UVM
3. Evolution and Milestones
4. Core Principles and Architecture
5. Cocotb (makes python understand hardware as different than software)
6. pyUVM
7. Simple pyUVM Example
8. Wrap Up Next Steps
9. Bibliography

UVM Core purpose

1. Reusability
2. Modularity
3. Scalability
4. Interoperability
5. Constrained Random Stimulus Generation
6. Functional Coverage
7. Separation of Concerns

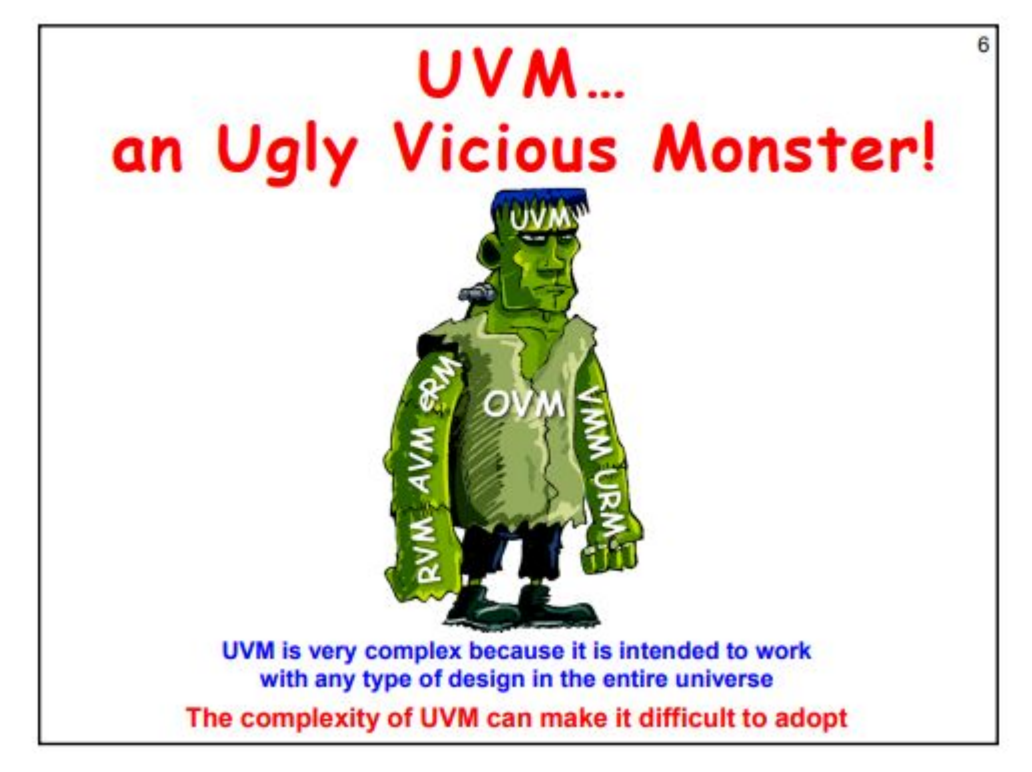
Origins





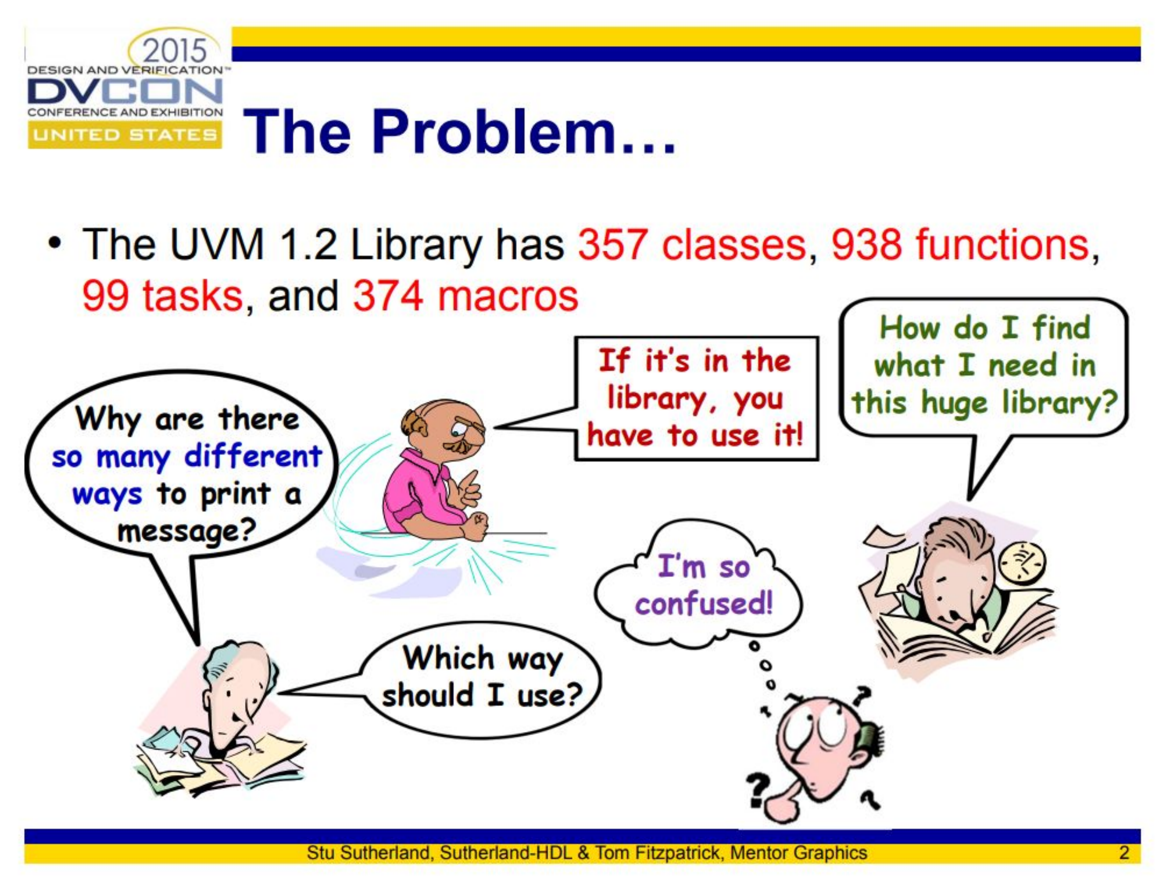
https://sutherland-hdl.com/papers/2016-DVClub-PDX_adopting_uvm_seminar.pdf

Reality



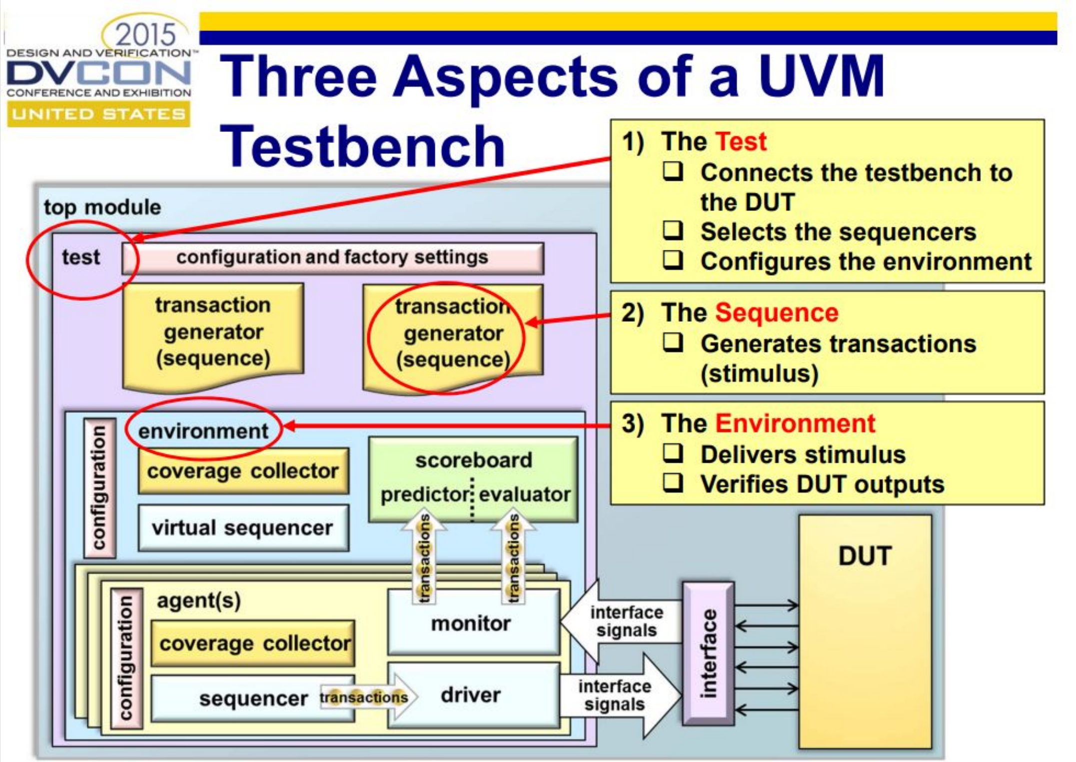
https://sutherland-hdl.com/papers/2016-DVClub-PDX_adopting_uvm_seminar.pdf

UVM



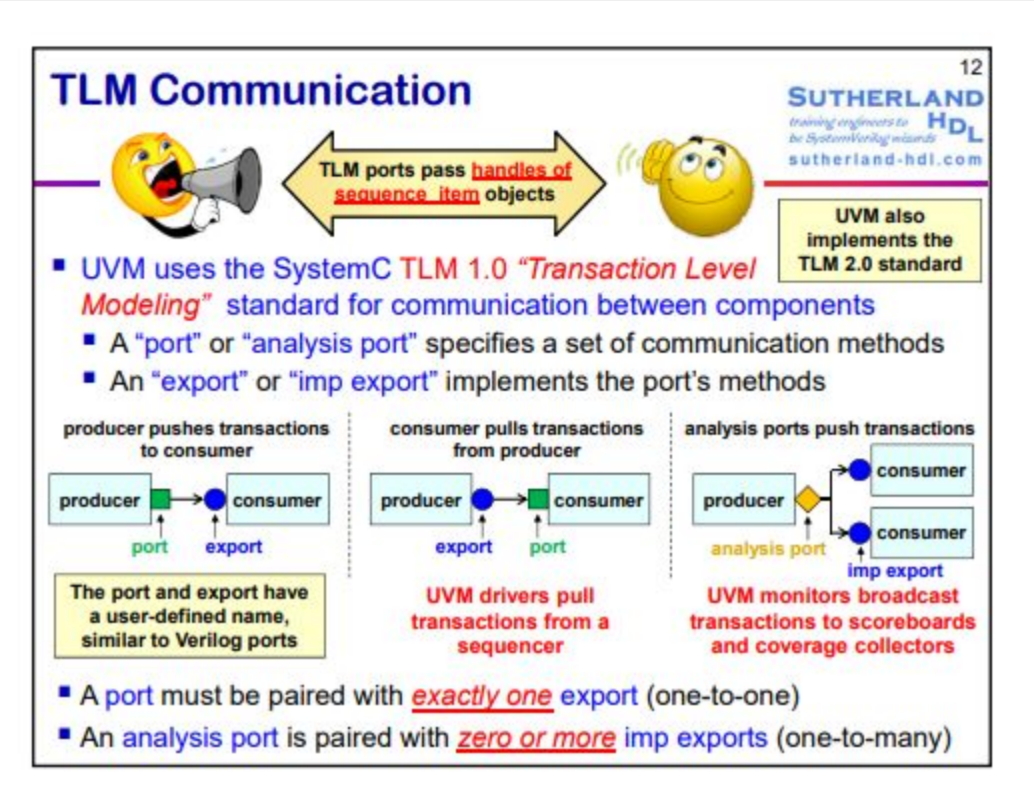
2015 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION UNITED STATES
The Problem…
• The UVM 1.2 Library has 357 classes, 938 functions, 99 tasks, and 374 macros
Why are there so many different ways to print a message?
If it's in the library, you have to use it!
How do I find what I need in this huge library?
I'm so confused!
Which way should I use?
Stu Sutherland, Sutherland-HDL & Tom Fitzpatrick, Mentor Graphics
2

Aspects



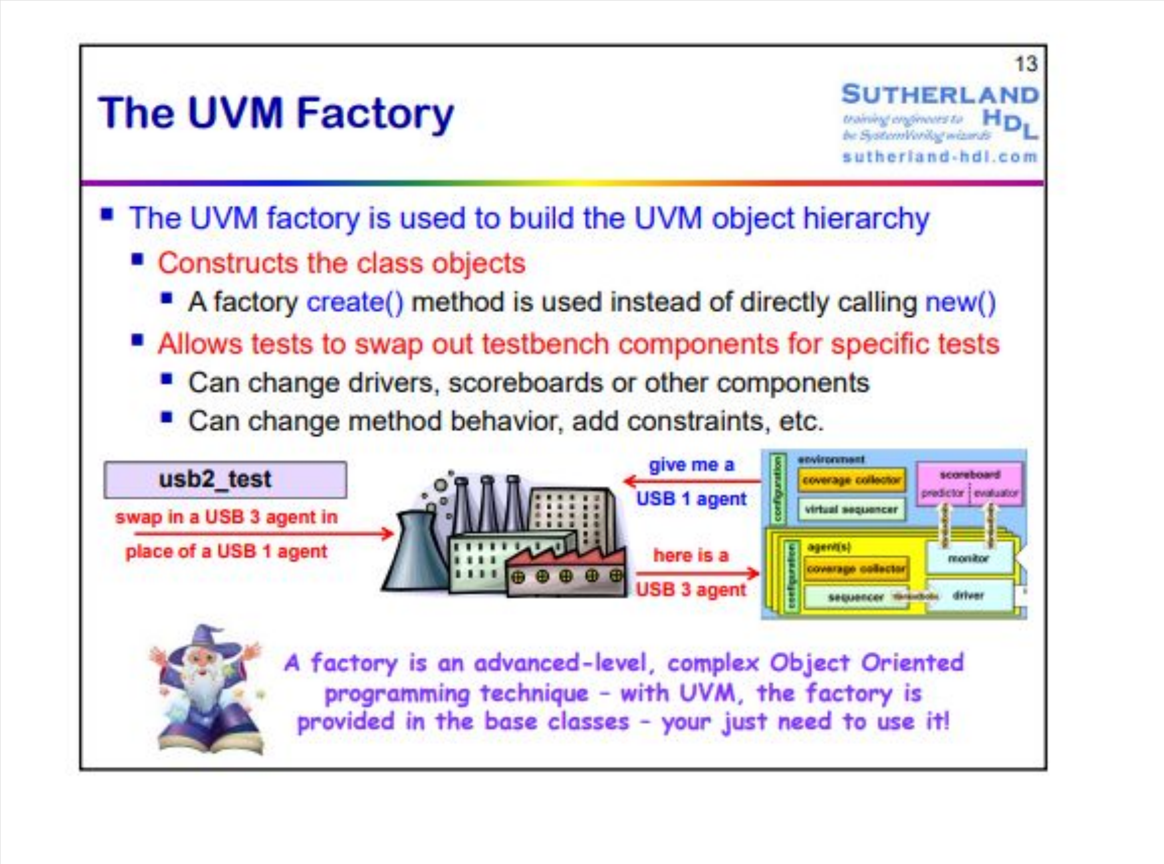
https://sutherland-hdl.com/papers/2015-DVCon_UVM-rapid-adoption_presentation.pdf

TLM



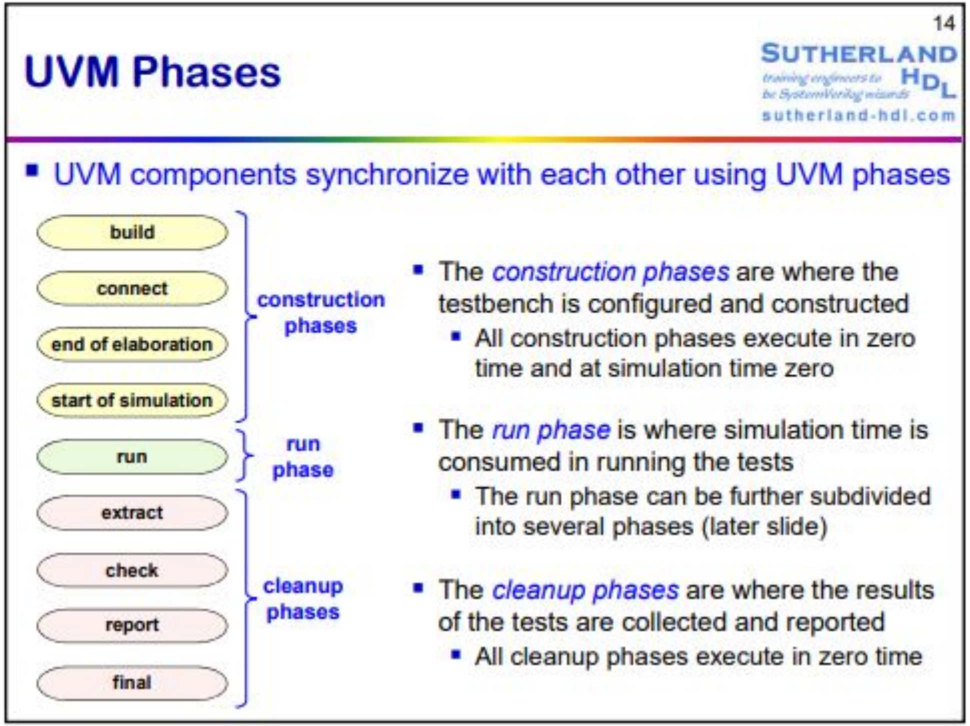
TLM Communication
12
SUTHERLAND HDL
training engineers to be SystemVerilog wizards
sutherland-hdl.com
TLM ports pass handles of sequence_item objects
UVM also implements the TLM 2.0 standard
UVM uses the SystemC TLM 1.0 "Transaction Level Modeling" standard for communication between components
A "port" or "analysis port" specifies a set of communication methods
An "export" or "imp export" implements the port's methods
producer pushes transactions to consumer
producer
consumer
port
export
The port and export have a user-defined name, similar to Verilog ports
consumer pulls transactions from producer
producer
consumer
export
port
UVM drivers pull transactions from a sequencer
analysis ports push transactions
producer
consumer
consumer
analysis port
imp export
UVM monitors broadcast transactions to scoreboards and coverage collectors
A port must be paired with exactly one export (one-to-one)
An analysis port is paired with zero or more imp exports (one-to-many)

UVM Factory



13
The UVM Factory
SUTHERLAND HDL
training engineers to be SystemVerilog wizards
sutherland-hdl.com
The UVM factory is used to build the UVM object hierarchy
Constructs the class objects
A factory create() method is used instead of directly calling new()
Allows tests to swap out testbench components for specific tests
Can change drivers, scoreboards or other components
Can change method behavior, add constraints, etc.
usb2_test
swap in a USB 3 agent in place of a USB 1 agent
give me a USB 1 agent
here is a USB 3 agent
environment
coverage collector
scoreboard
predictor
evaluator
virtual sequencer
agent(s)
coverage collector
monitor
sequencer
driver
A factory is an advanced-level, complex Object Oriented programming technique – with UVM, the factory is provided in the base classes – your just need to use it!

UVM Phases


14
UVM Phases
SUTHERLAND HDL
training engineers to be SystemVerilog wizards
sutherland-hdl.com
UVM components synchronize with each other using UVM phases
build
connect
end of elaboration
start of simulation
run
extract
check
report
final
construction phases
run phase
cleanup phases
The construction phases are where the testbench is configured and constructed
All construction phases execute in zero time and at simulation time zero
The run phase is where simulation time is consumed in running the tests
The run phase can be further subdivided into several phases (later slide)
The cleanup phases are where the results of the tests are collected and reported
All cleanup phases execute in zero time

UVM Phases

UVM Phase Tasks and Functions

15

SUTHERLAND HDL
training engineers to be SystemVerilog wizards
sutherland-hdl.com

- Each UVM component can start activity in any of the phases

```
class my_agent extends uvm_agent;
  ...
  function void build_phase(...);
    ... // construct components
  endfunction

  function void connect_phase(...);
    ... // connect components
  endfunction
endclass: my_agent
```

```
class my_driver extends uvm_driver;
  ...
  task run_phase(...);
    ... // get transaction & drive
  endfunction
endclass: my_driver
```

```
class my_sb extends uvm_scoreboard;
  ...
  function void build_phase(...);
    ... // construct components
  endfunction

  function void connect_phase(...);
    ... // connect components
  endfunction

  task run_phase(...);
    ... // evaluate dut outputs
  endtask

  function void report_phase(...);
    ... // report the pass/fail score
  endfunction
endclass: my_sb
```

Each phase does not end until all activity for that phase type has completed in every component (e.g.: *all* build phases must complete before any connect phase can start)

Cocotb

What is cocotb?

cocotb is a COroutine based COsimulation TestBench environment for verifying VHDL and SystemVerilog RTL using Python.

cocotb is completely free, open source (under the BSD License) and hosted on GitHub.

cocotb requires a simulator to simulate the HDL design and has been used with a variety of simulators on Linux, Windows and macOS.

How is cocotb different?

Encourages the same philosophy of design re-use and randomized testing as UVM, however is implemented in Python.

With cocotb, VHDL or SystemVerilog are normally only used for the design itself, not the testbench.

cocotb has built-in support for integrating with continuous integration systems, such as Jenkins, GitLab, etc. through standardized, machine-readable test reporting formats.

cocotb was specifically designed to lower the overhead of creating a test.

cocotb automatically discovers tests so that no additional step is required to add a test to a regression.

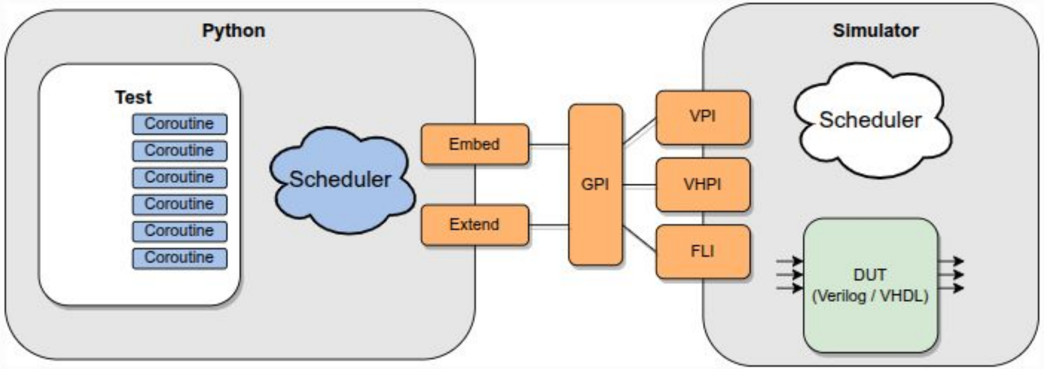
All verification is done using Python which has various advantages over using SystemVerilog or VHDL for verification:

- Writing Python is fast - it's a very productive language.
- It's easy to interface to other languages from Python.
- Python has a huge library of existing code to re-use.
- Python is interpreted - tests can be edited and re-run without having to recompile the design.
- Python is popular - far more engineers know Python than SystemVerilog

How Does it work?

How does cocotb work?

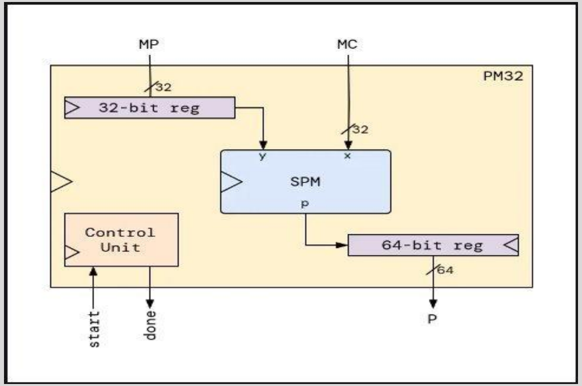
A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python. Note that cocotb can not instantiate HDL blocks - your DUT must be complete.



A test is simply a Python function. At any given time either the simulator is advancing time or the Python code is executing. The `await` keyword is used to indicate when to pass control of execution back to the simulator. A test can spawn multiple coroutines, allowing for independent flows of execution.

https://docs.cocotb.org/en/stable/

PM32 verif With Cocotb




```python
# test_my_design.py (extended)      wallo, 7 months ago • adding cocotb tb and pyuvm test, only test_my_d…

import cocotb
from cocotb.triggers import FallingEdge, Timer
from cocotb.types import Bit,Logic, LogicArray


async def generate_clock(dut):
    """Generate clock pulses."""

    for cycle in range(300):
        dut.clk.value = 0
        await Timer(1, units="ns")
        dut.clk.value = 1
        await Timer(1, units="ns")

async def reset_dut(dut, duration_ns):
    """Start with reset asserted and then de-assert it"""
    dut.rst.value = 1
    await Timer(duration_ns, units="ns")
    dut.rst.value = 0
    dut.rst._log.debug("Reset complete")


@cocotb.test()
async def my_first_test(dut):

    await cocotb.start(generate_clock(dut))  # run the clock "in the background"
    await cocotb.start(reset_dut(dut, 3))    # run reset in the background after 3 cycles get out of reset

    dut.start.value = 0
    dut.mc.value = 0
    dut.mp.value = 0
    await Timer(5, units="ns")  # wr 10
    dut.start.value = 1
    dut.mp.value = 16
    dut.mc.value = 16

    await Timer(2, units="ns")  # wr 10
    dut.start.value = 0

    await Timer(150, units="ns")  # wait for multiplication to complete
    await FallingEdge(dut.clk)  # wait for falling edge/"negedge"

    assert int(dut.p.value) == 16*16, "p is not 16*16!"
```

pyUVM

pyuvm is the Universal Verification Methodology implemented in Python instead of SystemVerilog. pyuvm uses cocotb to interact with simulators and schedule simulation events.

pyuvm takes advantage of Python's ease of use and object-oriented power to implement the most-often used parts of the IEEE 1800.2 standard.

It is easier to write UVM code in Python because Python does not have strict typing and does not require parameterized classes.

Python also supports important object-oriented programming (OOP) concepts such as multiple-inheritance that are missing in SystemVerilog.

https://pyuvm.github.io/pyuvm/docsources/README.html

There is Hope

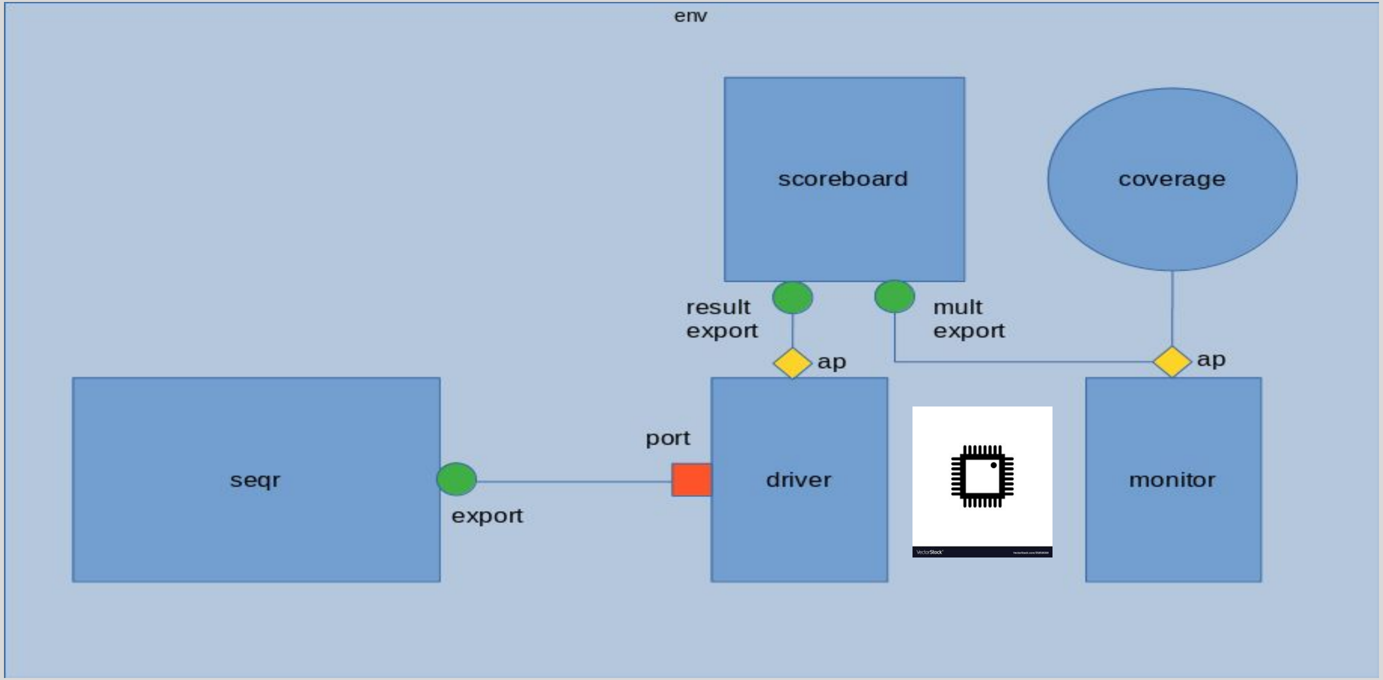
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

The Solution…

- The UVM 1.2 Library has 357 classes, 938 functions, 99 tasks, and 374 macros

- Our recommended subset in the paper uses 11 classes, 33 tasks/functions and 7 macros
- You really only need to learn 3% of UVM to be productive!
  - 2% of classes
  - 3% of methods

Plan


env
scoreboard
coverage
result
export
mult
export
ap
ap
port
seqr
driver
monitor
export

pyUVM

pyuvm import

```
from pyuvm import *          wallo, 7 months ago • adding cocotb tb an
import pyuvm
from cocotb.triggers import ClockCycles
import random
# All testbenches use tb_utils, so store it in a central
# place and add its path to the sys path so we can import it
import sys
from pathlib import Path
#sys.path.insert(0, str(Path("..").resolve()))
from tb_utils import DutBfm, Ops, DutPrediction  # noqa: E402
```

pyUVM

Driver

```python
# # UVM sequences
# ## Driver

# Edit: The Driver refactored to work with sequences
wallo, 7 months ago | 1 author (wallo)
class Driver(uvm_driver):                    (parameter) self: Self@Driver
    def start_of_simulation_phase(self):
        self.bfm = DutBfm()

    async def run_phase(self):
        await self.bfm.reset()
        self.bfm.start_tasks()
        while True:
            cmd = await self.seq_item_port.get_next_item()
            await self.bfm.send_op(cmd.mc, cmd.mp, cmd.op)
            self.seq_item_port.item_done()
```

pyUVM

Building the UVM TB

Instance the components

Connect the components

```
# ## Connecting the driver to the sequencer
wallo, 7 months ago | 1 author (wallo)
class DutEnv(uvm_env):

    # Edit: Instantiating the sequencer in the environment
    def build_phase(self):
        self.seqr = uvm_sequencer("seqr", self)
        ConfigDB().set(None, "*", "SEQR", self.seqr)
        self.driver = Driver("driver", self)
        self.cmd_mon = Monitor("cmd_mon", self, "get_cmd")
        self.result_mon = Monitor("result_mon", self, "get_result")
        self.scoreboard = Scoreboard("scoreboard", self)
        self.coverage = Coverage("coverage", self)

    # Edit: Connecting the sequencer to the driver
    def connect_phase(self):
        self.driver.seq_item_port.connect(self.seqr.seq_item_export)
        self.cmd_mon.ap.connect(self.scoreboard.cmd_export)
        self.cmd_mon.ap.connect(self.coverage.analysis_export)
        self.result_mon.ap.connect(self.scoreboard.result_export)
```

pyUVM

Define a sequence item

```python
# ## AluSeqItem
# Edit: Defining an ALU command as a sequence item
wallo, 7 months ago | 1 author (wallo)
class AluSeqItem(uvm_sequence_item):

    def __init__(self, name, aa, bb, op):
        super().__init__(name)
        self.mc = aa
        self.mp = bb
        self.op = Ops(op)

    # Edit: The __eq__ and __str__ methods in a sequence item
    def __eq__(self, other):
        same = self.mc == other.mc and self.mp == other.mp and self.op == other.op
        return same

    def __str__(self):
        return f"{self.get_name()} : mc: 0x{self.mc:02x} \
        OP: {self.op.name} ({self.op.value}) mp: 0x{self.mp:02x}"
```

pyuvm

Create interesting sequences

```python
# ## Creating sequences
# ### BaseSeq

# Edit: The BaseSeq contains the common body() method
wallo, 7 months ago | 1 author (wallo)
class BaseSeq(uvm_sequence):

    async def body(self):
        for op in list(Ops):
            cmd_tr = AluSeqItem("cmd_tr", 0, 0, op)
            await self.start_item(cmd_tr)
            self.set_operands(cmd_tr)
            await self.finish_item(cmd_tr)

    def set_operands(self, tr):
        pass


# ### RandomSeq and MaxSeq
# Edit: Extending BaseSeq to create the random and maximum stimulus
wallo, 7 months ago | 1 author (wallo)
class RandomSeq(BaseSeq):
    def set_operands(self, tr):
        tr.mc = random.randint(0, (2**32 -1))
        tr.mp = random.randint(0, (2**32 -1))


wallo, 7 months ago | 1 author (wallo)
class MaxSeq(BaseSeq):
    def set_operands(self, tr):
        tr.mc = 0xffff_ffff
        tr.mp = 0xffff_ffff
```

pyvm

Create tests based on

sequences

```
# ## Starting a sequence in a test
# ### BaseTest

# Edit: All tests use the same environment and need a sequence
wallo, 7 months ago | 1 author (wallo)
@pyuvm.test()
class BaseTest(uvm_test):
    def build_phase(self):
        self.env = DutEnv("env", self)

    def end_of_elaboration_phase(self):
        self.seqr = ConfigDB().get(self, "", "SEQR")

    # Edit: All tests start the sequence
    async def run_phase(self):
        self.raise_objection()
        seq = BaseSeq.create("seq")
        await seq.start(self.seqr)
        await ClockCycles(cocotb.top.clk, 200)  # to do last transaction
        self.drop_objection()


# ### RandomTest and MaxTest
# Edit: Overriding BaseSeq to get random stimulus and all ones
wallo, 7 months ago | 1 author (wallo)
@pyuvm.test()
class RandomTest(BaseTest):
    def start_of_simulation_phase(self):
        uvm_factory().set_type_override_by_type(BaseSeq, RandomSeq)


wallo, 7 months ago | 1 author (wallo)
@pyuvm.test()
class MaxTest(BaseTest):
    def start_of_simulation_phase(self):
        uvm_factory().set_type_override_by_type(BaseSeq, MaxSeq)
```

pyuvm

coverage

```
wallo, 7 months ago | 1 author (wallo)
class Coverage(uvm_subscriber):

    def end_of_elaboration_phase(self):
        self.cvg = set()

    def write(self, cmd):
        (_, _, op) = cmd
        self.cvg.add(op)

    def report_phase(self):
        if len(set(Ops) - self.cvg) > 0:
            self.logger.error(
                f"Functional coverage error. Missed: {set(Ops)-self.cvg}")
            assert False
        else:
            self.logger.info("Covered all operations")
            assert True
```

pyuvm

scoreboard

```
wallo, 7 months ago | 1 author (wallo)
class Scoreboard(uvm_component):

    def build_phase(self):
        self.cmd_fifo = uvm_tlm_analysis_fifo("cmd_fifo", self)
        self.result_fifo = uvm_tlm_analysis_fifo("result_fifo", self)
        self.cmd_get_port = uvm_get_port("cmd_get_port", self)
        self.result_get_port = uvm_get_port("result_get_port", self)
        self.cmd_export = self.cmd_fifo.analysis_export
        self.result_export = self.result_fifo.analysis_export

    def connect_phase(self):
        self.cmd_get_port.connect(self.cmd_fifo.get_export)
        self.result_get_port.connect(self.result_fifo.get_export)

    def check_phase(self):
        while self.result_get_port.can_get():
            _, actual_result = self.result_get_port.try_get()
            cmd_success, cmd = self.cmd_get_port.try_get()
            if not cmd_success:
                self.logger.critical(f"result {actual_result} had no command")
            else:
                (A, B, op_numb) = cmd
                op = Ops(op_numb)
                predicted_result = DutPrediction(A, B, op)
                if predicted_result == actual_result:
                    self.logger.info(f"PASSED: 0x{A:02x} {op.name} 0x{B:02x} ="
                                     f" 0x{actual_result:04x}")
                else:
                    self.logger.error(f"FAILED: 0x{A:02x} {op.name} 0x{B:02x} "
                                      f"= 0x{actual_result:04x} "
                                      f"expected 0x{predicted_result:04x}")
```

uvm

monitor

wallo, 7 months ago | 1 author (wallo)

```python
class Monitor(uvm_component):
    def __init__(self, name, parent, method_name):
        super().__init__(name, parent)
        self.bfm = DutBfm()
        self.get_method = getattr(self.bfm, method_name)

    def build_phase(self):
        self.ap = uvm_analysis_port("ap", self)

    async def run_phase(self):
        while True:
            datum = await self.get_method()
            self.logger.debug(f"MONITORED {datum}")
            self.ap.write(datum)
```

Run example

Conclusions

UVM was created by combining approaches from multiple vendors

UVM is relatively hard to start with, as its flexibility make it extensive.

- However one can focus on a subset to start with.
- It is key to be familiarized with OOP concepts
- It is key to be familiarized with monitor, scoreboard, env, sequencer, driver, sequence, etc

One Suggestion is to split the UVM work in 3 roles:

- Environment Writer
- Sequence Writer
- Test Writes

Learnings

Cocotb: python to create simple TB.

pyUVM: allows for scalable, professional solutions.

Functional Verification: is an entire world, no wonder why engineers work on this and specialize in specific sub-areas.

Divide and conquer approach can be used on complex areas.

Gemini research considerably helped to “synthesize” a lot of information from different source to generate this material

There is a lot of experts in the world that have gone thru this and have shared their learnings in conferences. (Gurus from Sunburst, Sutherland, DV Con, Siemens, etc)

1. FPGA Verification, accessed May 13, 2025, https://verificationacademy.com/topics/fpga-verification/
2. Agile Development Methodologies for Hardware Verification and Validation - Valispace, accessed May 13, 2025, https://www.valispace.com/agile-dev-methodologies-harware-verification-validation/
3. Universal Verification Methodology - Wikipedia, accessed May 13, 2025, https://en.wikipedia.org/wiki/Universal_Verification_Methodology
4. Emulators and Debuggers in Embedded System, OVM UVM, accessed May 13, 2025, https://www.aldec.com/en/solutions/functional_verification/uvm_ovm_vmm--emulators-and-debuggers-in-embedded-system
5. Understanding Verification Methodologies: OVM, UVM, and VMM : r ..., accessed May 13, 2025, https://www.reddit.com/r/vlsi_enthusiast/comments/1e65tn6/understanding_verification_methodologies_ovm_uvm/
6. Accellera - Wikipedia, accessed May 13, 2025, https://en.wikipedia.org/wiki/Accellera
7. The Early History of Accellera, accessed May 13, 2025, https://www.accellera.org/about/the-early-history-of-accellera
8. Tutorial: Accellera-UVM-Tutorial-2013.pdfLessons from the ..., accessed May 13, 2025, https://www.accellera.org/resources/videos/uvm-tutorial-2013
9. Introducing IEEE 1800.2 The Next Step for UVM - Accellera Systems Initiative, accessed May 13, 2025, https://www.accellera.org/images/resources/videos/Accellera-UVM-Tutorial-2017.pdf
10. UVM KnowHow - Doulos, accessed May 13, 2025, https://www.doulos.com/knowhow/systemverilog/uvm/

1. Download UVM (Universal Verification Methodology), accessed May 13, 2025, https://www.accellera.org/downloads/standards/uvm
2. IEEE 1800.2-2017 - Accuris Standards Store, accessed May 13, 2025, https://store.accuristech.com/ieee/products/vendor_id/6021
3. Accellera Board Approves Universal Verification Methodology for Mixed-Signal (UVM-MS) 1.0 Standard for Release, accessed May 13, 2025, https://www.accellera.org/news/press-releases/408-accellera-board-approves-universal-verification-methodology-for-mixed-signal-uvm-ms-1-0-standard-for-release
4. Accellera Board Approves Universal Verification Methodology for Mixed-Signal (UVM-MS) 1.0 Standard for Release - GlobeNewswire, accessed May 13, 2025, https://www.globenewswire.com/news-release/2025/02/04/3020562/0/en/Accellera-Board-Approves-Universal-Verification-Methodology-for-Mixed-Signal-UVM-MS-1-0-Standard-for-Release.html
5. UVM - Universal Verification Methodology | Siemens Verification ..., accessed May 13, 2025, https://verificationacademy.com/topics/uvm-universal-verification-methodology/
6. Reusability of the testbench - UVM - Verification Academy, accessed May 13, 2025, https://verificationacademy.com/forums/t/reusability-of-the-testbench/45916
7. Typical UVM Testbench Architecture | The Art Of Verification, accessed May 13, 2025, https://theartofverification.com/uvm-testbench-architecture/
8. Introduction to the UVM | Kasun Buddhi, accessed May 13, 2025, https://kasunbuddhi.com/blogs/introduction-to-the-uvm/
9. UVM - Step by Step guide to learning UVM . | VLSI Interview questions - Skillshare, accessed May 13, 2025,

1. UVM Monitor - VLSI Verify, accessed May 13, 2025, https://vlsiverify.com/uvm/uvm-monitor/
2. UVM Scoreboard - VLSI Verify, accessed May 13, 2025, https://vlsiverify.com/uvm/uvm-scoreboard/
3. github.com, accessed May 13, 2025, https://github.com/pyuvm/pyuvm#:~:text=pyuvm%20is%20the%20Universal%20Verification,of%20the%20IEEE%201800.2%20standard.
4. www.syosil.com, accessed May 13, 2025, https://www.syosil.com/images/resources/osv_whitepaper-1.0.3.0.pdf
5. pyuvm/pyuvm: The UVM written in Python - GitHub, accessed May 13, 2025, https://github.com/pyuvm/pyuvm
6. Python and the UVM - Verification Horizons, accessed May 13, 2025, https://blogs.sw.siemens.com/verificationhorizons/2021/09/09/python-and-the-uvm/
7. indico.fnal.gov, accessed May 13, 2025, https://indico.fnal.gov/event/64625/contributions/295304/attachments/179495/245171/FC_Presentation.pdf
8. Transitioning from SV/UVM to pyUVM: A Practical Guide – Part 2 - ben-haroosh, accessed May 13, 2025, https://benharoosh.co.il/transitioning-from-sv-uvm-to-pyuvm-part-2/
9. hurisson/pyuvm_primer: Examples for using pyuvm - GitHub, accessed May 13, 2025, https://github.com/hurisson/pyuvm_primer

1. UVM testbench generator : r/chipdesign - Reddit, accessed May 13, 2025, https://www.reddit.com/r/chipdesign/comments/1abutgc/uvm_testbench_generator/
2. The configuration database in pyuvm - Verification Horizons, accessed May 13, 2025, https://blogs.sw.siemens.com/verificationhorizons/2021/10/27/the-configuration-database-in-pyuvm/
3. My Blog – Chen Ben-Haroosh, accessed May 13, 2025, https://benharoosh.co.il/blog/
4. Python for RTL Verification - Google Sites, accessed May 13, 2025, https://sites.google.com/view/python-4-rtl-verif/home
5. Python for RTL Verification: A complete course in Python, cocotb, and pyuvm - Amazon.sg, accessed May 13, 2025, https://www.amazon.sg/Python-RTL-Verification-complete-course/dp/B0BCZ1JM3P
6. https://sutherland-hdl.com/papers/2016-DVClub-PDX_adopting_uvm_seminar.pdf
7. https://sutherland-hdl.com/papers/2015-DVCon_UVM-rapid-adoption_presentation.pdf