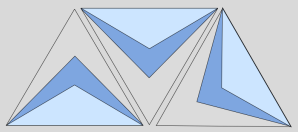
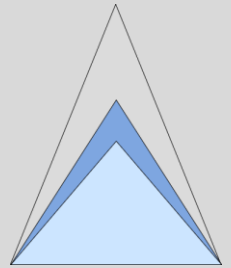
AVL

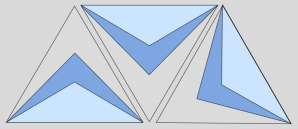
Bringing Industry Best Practice Test-Bench Design to Open Source

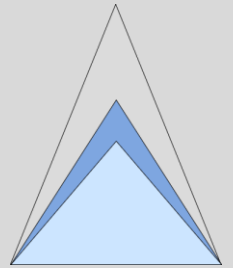




What is AVL?

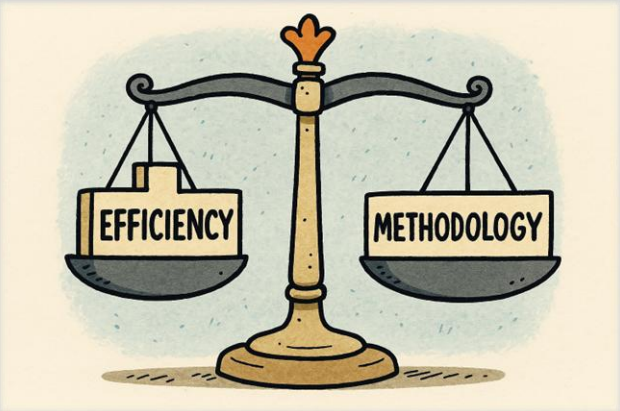
- The Apheleia Verification Library is an open-source python library
- AVL is built on CocoTB
  - Universal simulator support
  - Active user community
  - Near Zero compile time overhead





Why Use AVL?

- AVL is not a UVM implementation in python
- AVL is not a minimal test-bench language
- AVL takes combines the re-use best practices of UVM and efficiency of python
- AVL is an engineer driven test-bench library enabling scalable verification environments with a focus on productivity – not methodology

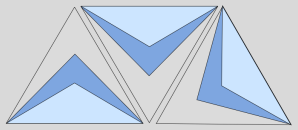


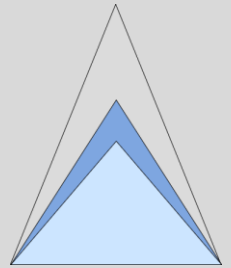
Who is AVL Aimed At?

- AVL is aimed at designers and verification engineers
  - Novices and students
  - Industry experts
  - Hobbyists
  - Professionals
- Anyone who wants to spend more time doing verification and less time developing code

AVL Features

- HDL centric variables
- Constrained Random
- Familiar methodology
  - Sequences
  - Drivers
  - Agents
- Familiar re-use
  - Factory
  - Phases
  - TLM style ports
- Functional Coverage
  - Run-time defined
- Statistical Coverage
- Visualization
- Multi-purpose logging
  - Human Readable
  - Machine Readable



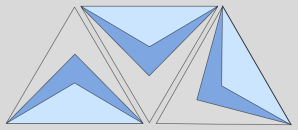


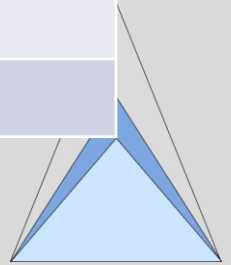
AVL Variables

- Pythons "Duck Typing" makes for easy coding and re-use but doesn't naturally fit with HDL
  - But it does mean you don't need to parameterize classes unnecessarily
- AVL supports python's native data types while providing a richer, hardware centric set

AVL Variables

System Verilog	Python	AVL
shortint	int	avl.Int16
int / integer	int	avl.Int32
longint	int	avl.Int64
byte	int	avl.Byte / avl.Int8
logic / bit	bool / int	avl.Logic / avl.Bool / avl.Uint<N>
time	int	avl.Int64
real	float	avl.Double / avl.Fp64
shortreal	float	avl.Half / avl.Fp16
	float	avl.Float / avl.Fp32
string	str	str
enum	Enum	avl.Enum

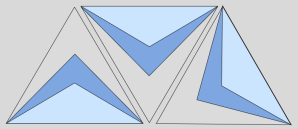


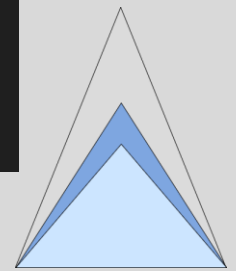


AVL Variables

- Once defined all AVL variables behave like python variables
  - Arithmetic operations
  - Comparison
- Wrapping and sign are handled naturally
- Each variable can have a defined string format for easier debug

```
a = avl.Int("a", 10, width=8)
b = avl.Int("b", 5, width=8)

# Arithmetic
assert a + b == 15              # 10 + 5 = 15
c = avl.Int("c", a, width=8)
c += b
assert c == 15                  # 10 + 5 = 15

assert a - b == 5               # 10 - 5 = 5
assert b - a == -5              # 5 - 10 = -5
c = avl.Int("c", b, width=8)
c -= a
assert c == -5                  # 5 - 10 = -5

assert a * b == 50              # 10 * 5 = 50
c = avl.Int("c", a, width=8)
c *= b
assert c == 50                  # 10 * 5 = 50

assert a // b == 2              # 10 // 5 = 2
c = avl.Int("c", a, width=8)
c //= b
assert c == 2                   # 10 // 5 = 2

assert a % b == 0               # 10 % 5 = 0
c = avl.Int("c", a, width=8)
c %= b
assert c == 0                   # 10 % 5 = 0

base = avl.Int("base", 2, width=8)
exp = avl.Int("exp", 4, width=8)
assert base ** exp == 16        # 2 ** 4 = 16
c = avl.Int("c", base, width=8)
c **= exp
assert c == 16                  # 2 ** 4 = 16

assert b << 1 == 10             # 5 << 1 = 10
c = avl.Int("c", b, width=8)
c <<= 1
assert c == 10                  # 5 << 1 = 10

assert a >> 1 == 5              # 10 >> 1 = 5
c = avl.Int("c", a, width=8)
c >>= 1
assert c == 5                   # 10 >> 1 = 5

# RDivMod
r = divmod(23, a)
assert r == (2, 3)              # 23 // 10 = 2, 23 % 10 = 3
r = divmod(a, 3)
assert r == (3, 1)              # 10 // 3 = 3, 10 % 3 = 1

# Comparisons
assert a > b                    # 10 > 5
assert not (a < b)
assert a >= b
assert not (a <= b)
assert a != b
assert a == avl.Int("a2", 10, width=8)
```

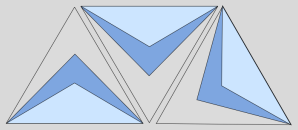


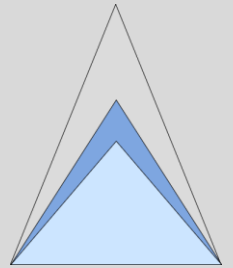


Constrained Random

- Flexible randomisation
  - Python Random or NumPy randomization

```
# Random variable
solution = random.choice([1, 2, 3])
```

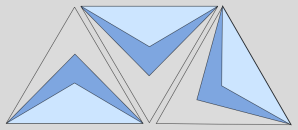


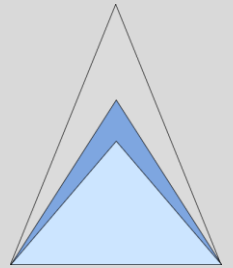


Constrained Random

- Flexible randomisation
  - Python Random or NumPy randomization
  - Python-Constraints library

```
# Create a constraint problem
problem = Problem()

# Add a variable with a domain (e.g., x can be 1, 2, or 3)
problem.addVariable("x", [1, 2, 3])

# Add a simple constraint (e.g., x must be greater than 1)
problem.addConstraint(lambda x: x > 1, ["x"])

# Solve the problem
solutions = problem.getSolutions()
```

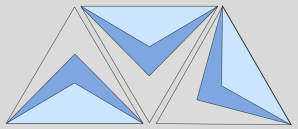


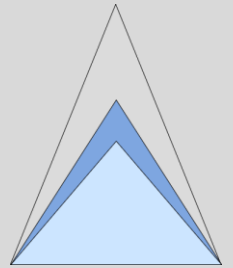


Constrained Random

- Flexible randomisation
  - Python Random or NumPy randomization
  - Python-Constraints library
  - Z3 constraints

```
self.a = avl.Logic("a", 0, width=8, fmt=hex)
self.b = avl.Logic("b", 0, width=8, fmt=hex)

self.add_constraint("c_0", lambda x: Or(x == 0, x == 100), self.a)
self.add_constraint("c_1", lambda x: And(x >= 5, x <= 100), self.b)
self.add_constraint("c_2", lambda x, y: Implies(x == 0, y == 10), self.a, self.b)
```

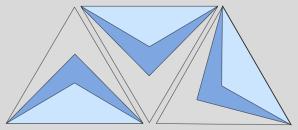


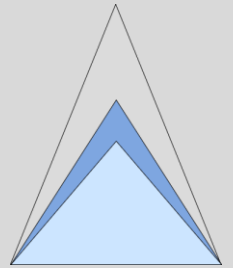


Z3 Constraints

- Z3 is an open-source constraint solver from Microsoft
- Supports bool, int, uint, enum and **float** numbers
- Handles wide variables
- Integrated into avl.Vars
  - Individual variable constraints
  - Multi-variable implication

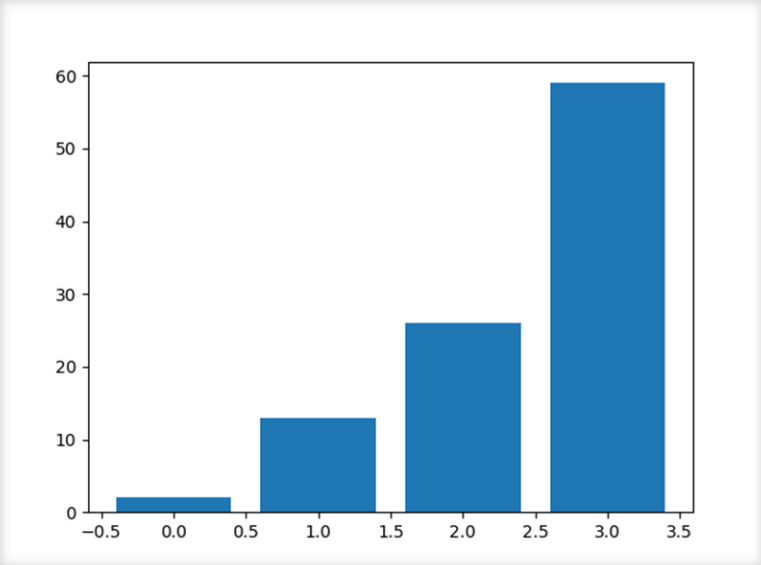
```
self.a = avl.Logic("a", 0, width=8, fmt=hex)
self.b = avl.Logic("b", 0, width=8, fmt=hex)

self.add_constraint("c_0", lambda x: Or(x == 0, x == 100), self.a)
self.add_constraint("c_1", lambda x: And(x >= 5, x <= 100), self.b)
self.add_constraint("c_2", lambda x, y: Implies(x == 0, y == 10), self.a, self.b)
```

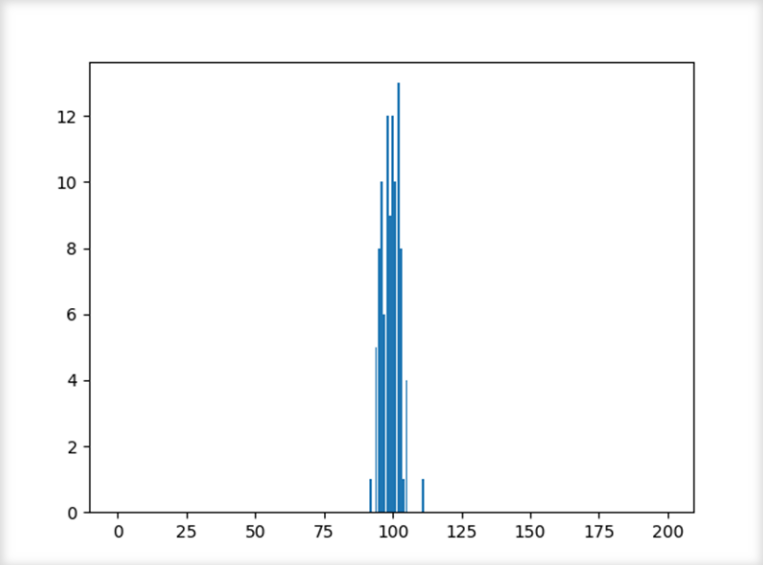




Distributions

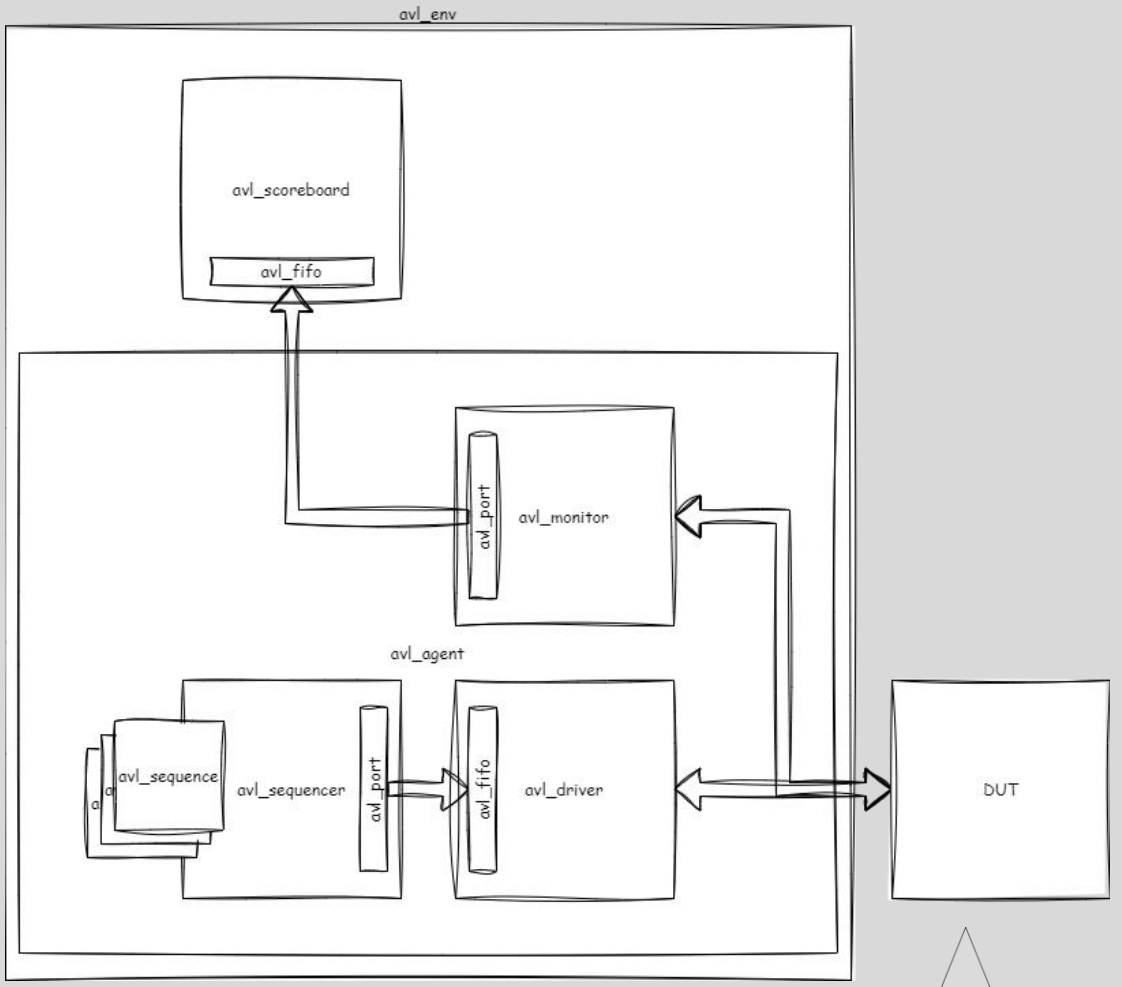
```
self.a = avl.Logic("a", 0, width=8, fmt=hex)
self.a.add_constraint(
    "d_0", lambda x: x == random.choices([0, 1, 2, 3], k=1, weights=[1, 2, 4, 8])[0]
)

self.b = avl.Logic("b", 0, width=32, fmt=hex)
self.b.add_constraint("d_1", lambda x: x == int(np.random.normal(100, 3)))
```





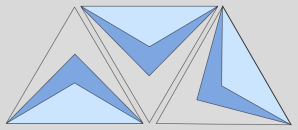
AVL Methodology

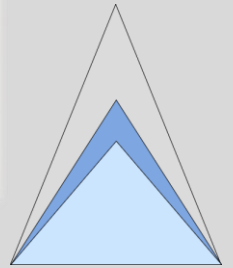
- AVL follows the UVM methodology
- Familiar and consistent
- No need for parameterization
- Direct access
  - No requirement for virtual interfaces



Scoreboards

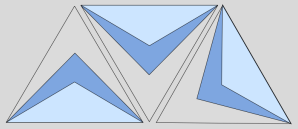
- Common mistakes weaken scoreboards
- Index scoreboard provides same API, but splits items into multiple in-order scoreboards based on an item attribute
- All scoreboards provide:
  - End-of-sim empty checks
  - Minimum check count

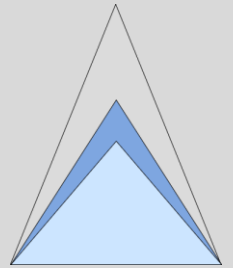




Testbench Helper Functions

- Environment class provides common testbench helper functions
  - Clock Generator defined by frequency
  - Sync / Async reset generators
  - Hard and Soft Timeouts
  - Tickers
- Vanilla template provided
  - Only need to override item type and implement driver / monitor run_phase



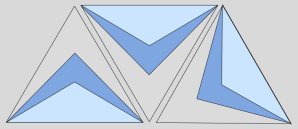


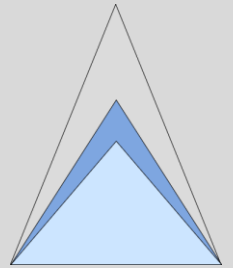
Factory

- AVL provides familiar factory features
  - Override by type and instance
  - Set / Get variables
- Built into __new__ method
  - No need for create()

```
avl_factory.set_override_by_instance('env.s', sub_comp_B)
avl_factory.set_override_by_instance('env.o', object_B)

e = example_env('env', None)
```

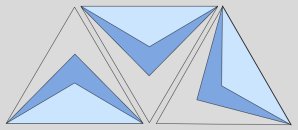
```
avl_factory.set_variable('env.a', 100)

e = example_env('env', None)
```

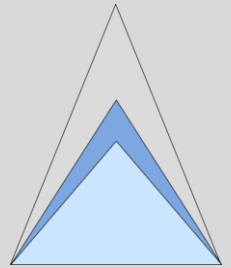




Attributes and Registration

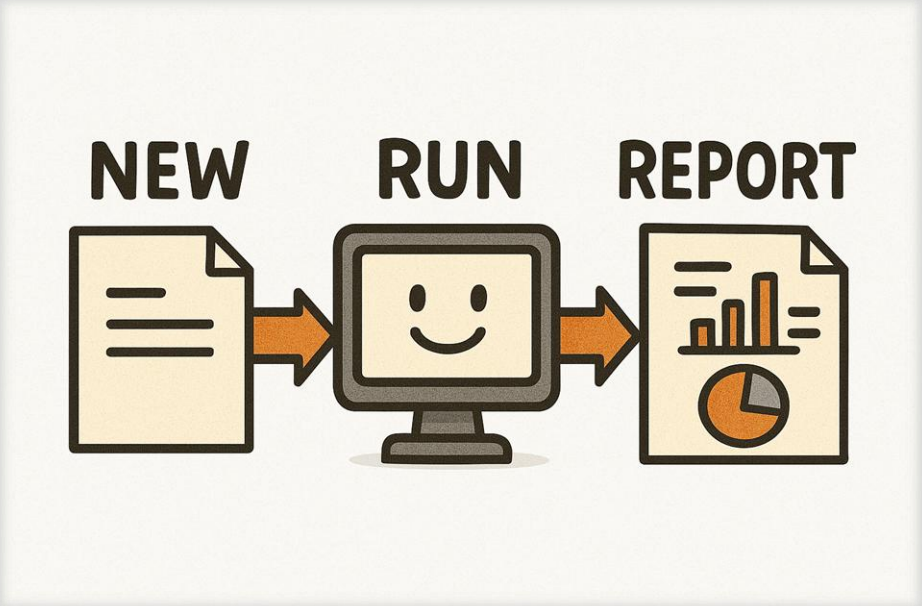
- NO MACROS
- All local variables automatically part of print and compare
  - Hidden variables identified by leading _ in variable name
- Copying should be done using Python’s built-in copy methods





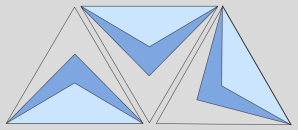
Phases

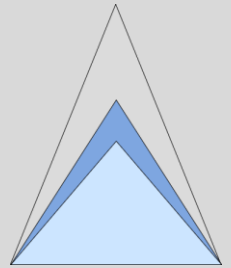
- AVL provides familiar phases and objection mechanism
- User can add, insert, remove and re-order phases
- Simplified – by default
  - Run Phase
  - Report Phase



Ports

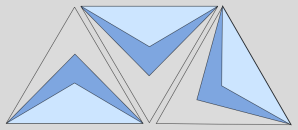
- Only 1 port always capable of broadcast
- avl.List extends native python list
  - Adds blocking_get() / blocking_pop()
- avl.Fifo extends avl.List
  - Adds blocking_push()
- Callbacks built into avl.Transaction not sequencer
  - Synchronisation on item-by-item
  - Simplifies sequences with multiple in-flight transactions or out-of-order responses



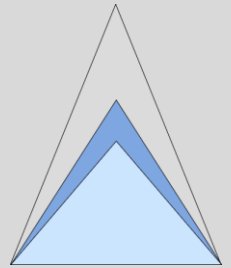


Functional Coverage

- AVL functional coverage familiar to SystemVerilog coverage
  - Covergroups, Coverpoints, Covercrosses
- Defined at run-time
- Flexible bin definition
  - Value in list, range or function
- Analysed and exported using Pandas
  - Simple, multi-sim merging and ranking

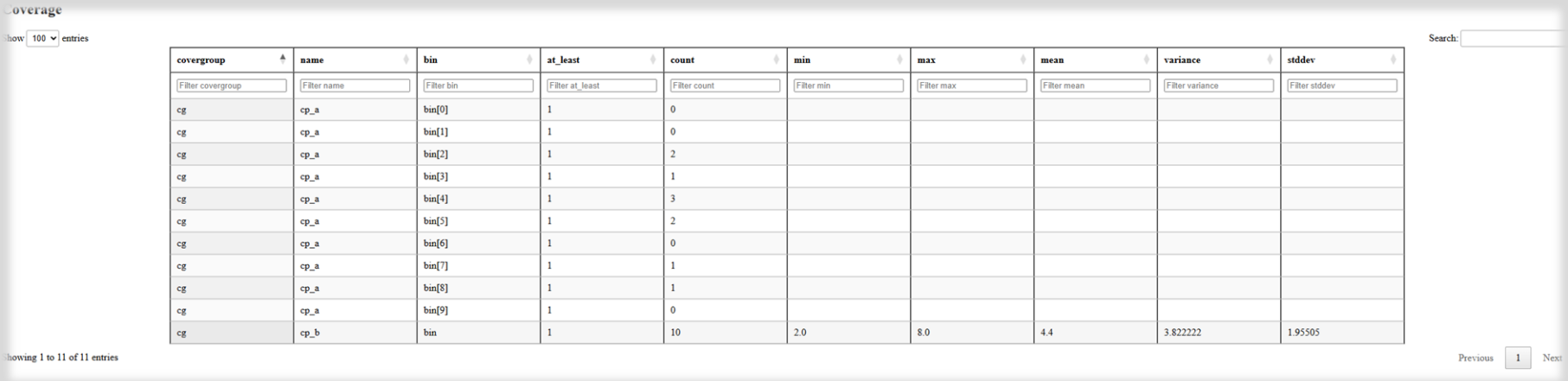


```
self.cg = avl.Covergroup("cg", self)
self.cp_a = self.cg.add_coverpoint("cp_a", lambda: self.a)
self.cp_a.add_bin("bin", 1, 2, range(100, 200))

self.cp_b = self.cg.add_coverpoint("cp_b", lambda: self.b)
self.cp_b.add_bin("bin", lambda x: x == 10)
```



Functional Coverage – Statistical Bins

- Bins can also collect statistics
  - Not just hit / unhit
  - Min, Max, Mean and StdDev of sampled values
- Provides native support for performance measurements



overage

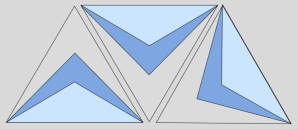
how 100 entries

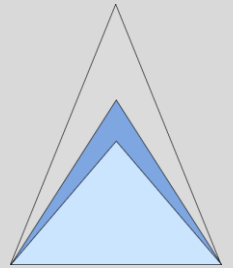
Search:

covergroup	name	bin	at_least	count	min	max	mean	variance	stddev
Filter covergroup	Filter name	Filter bin	Filter at_least	Filter count	Filter min	Filter max	Filter mean	Filter variance	Filter stddev
cg	cp_a	bin[0]	1	0					
cg	cp_a	bin[1]	1	0					
cg	cp_a	bin[2]	1	2					
cg	cp_a	bin[3]	1	1					
cg	cp_a	bin[4]	1	3					
cg	cp_a	bin[5]	1	2					
cg	cp_a	bin[6]	1	0					
cg	cp_a	bin[7]	1	1					
cg	cp_a	bin[8]	1	1					
cg	cp_a	bin[9]	1	0					
cg	cp_b	bin	1	10	2.0	8.0	4.4	3.822222	1.95505

howing 1 to 11 of 11 entries

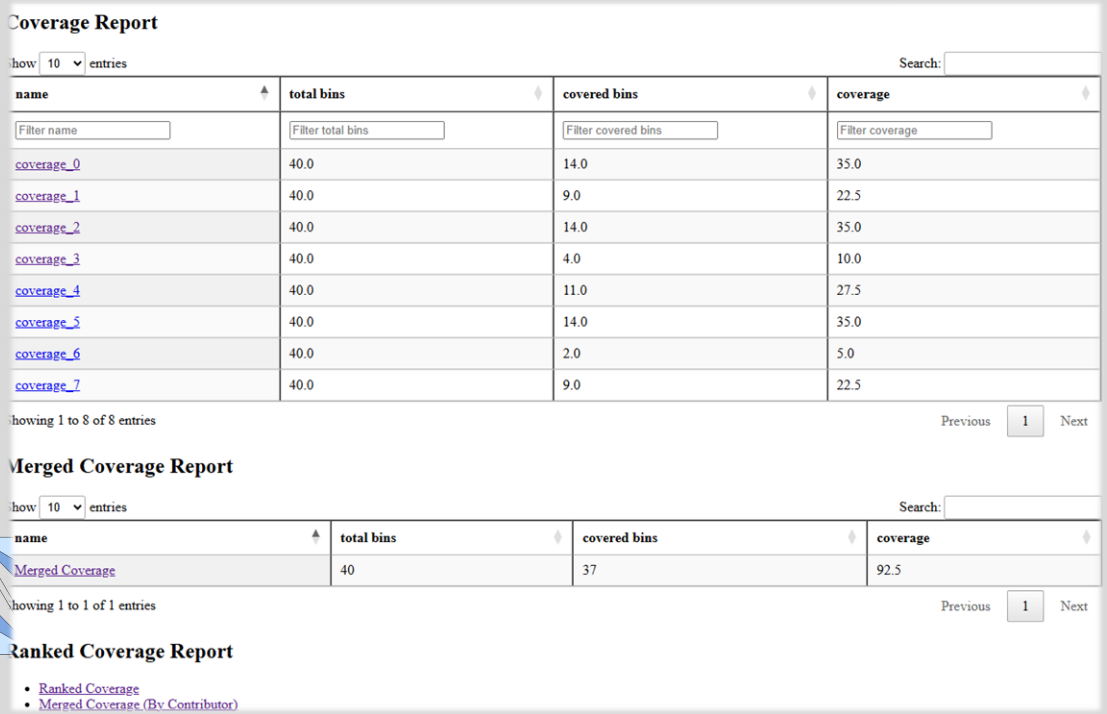
Previous 1 Next





Functional Coverage – Reporting

- Simple coverage reporting
  - Coverage Merging
  - Coverage Ranking
  - Regression optimization



Coverage Report

Show 10 entries Search:

name	total bins	covered bins	coverage
Filter name	Filter total bins	Filter covered bins	Filter coverage
coverage_0	40.0	14.0	35.0
coverage_1	40.0	9.0	22.5
coverage_2	40.0	14.0	35.0
coverage_3	40.0	4.0	10.0
coverage_4	40.0	11.0	27.5
coverage_5	40.0	14.0	35.0
coverage_6	40.0	2.0	5.0
coverage_7	40.0	9.0	22.5

Showing 1 to 8 of 8 entries Previous 1 Next

Merged Coverage Report

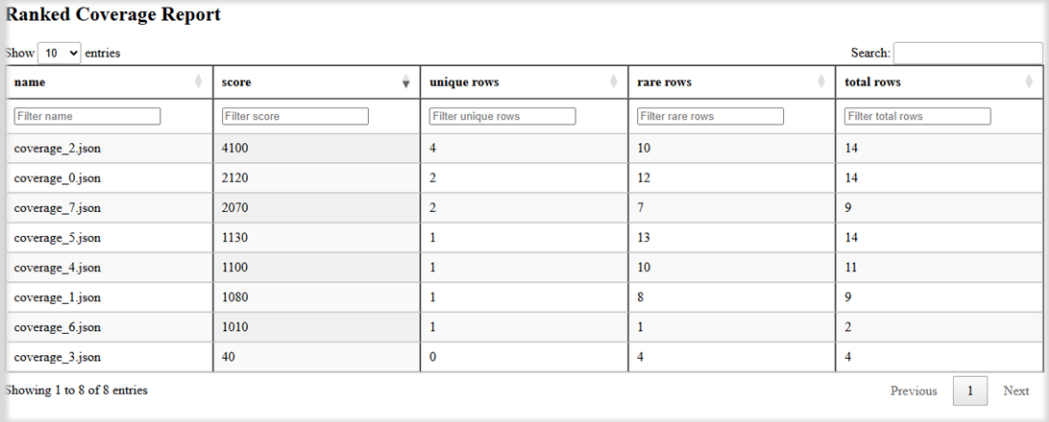
Show 10 entries Search:

name	total bins	covered bins	coverage
Merged Coverage	40	37	92.5

Showing 1 to 1 of 1 entries Previous 1 Next

Ranked Coverage Report

- Ranked Coverage
- Merged Coverage (By Contributor)

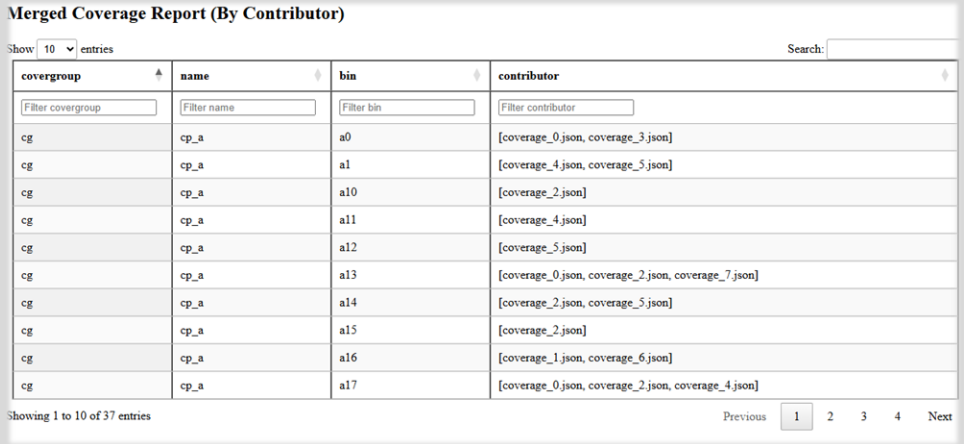


Ranked Coverage Report

Show 10 entries Search:

name	score	unique rows	rare rows	total rows
Filter name	Filter score	Filter unique rows	Filter rare rows	Filter total rows
coverage_2.json	4100	4	10	14
coverage_0.json	2120	2	12	14
coverage_7.json	2070	2	7	9
coverage_5.json	1130	1	13	14
coverage_4.json	1100	1	10	11
coverage_1.json	1080	1	8	9
coverage_6.json	1010	1	1	2
coverage_3.json	40	0	4	4

Showing 1 to 8 of 8 entries Previous 1 Next



Merged Coverage Report (By Contributor)

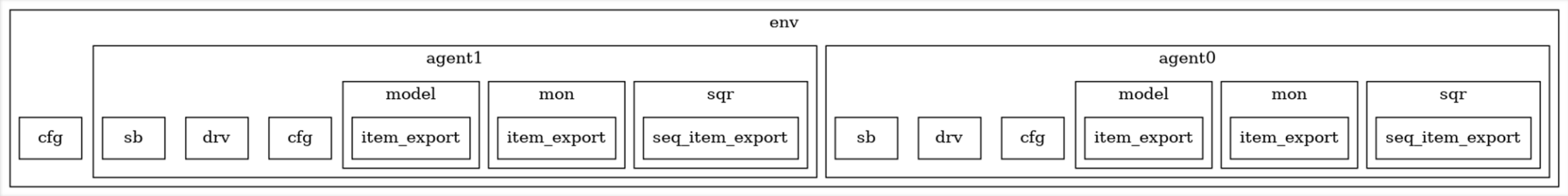
Show 10 entries Search:

covergroup	name	bin	contributor
Filter covergroup	Filter name	Filter bin	Filter contributor
cg	cp_a	a0	[coverage_0.json, coverage_3.json]
cg	cp_a	a1	[coverage_4.json, coverage_5.json]
cg	cp_a	a10	[coverage_2.json]
cg	cp_a	a11	[coverage_4.json]
cg	cp_a	a12	[coverage_5.json]
cg	cp_a	a13	[coverage_0.json, coverage_2.json, coverage_7.json]
cg	cp_a	a14	[coverage_2.json, coverage_5.json]
cg	cp_a	a15	[coverage_2.json]
cg	cp_a	a16	[coverage_1.json, coverage_6.json]
cg	cp_a	a17	[coverage_0.json, coverage_2.json, coverage_4.json]

Showing 1 to 10 of 37 entries Previous 1 2 3 4 Next

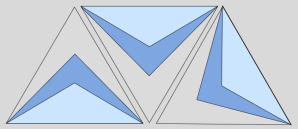
Visualization

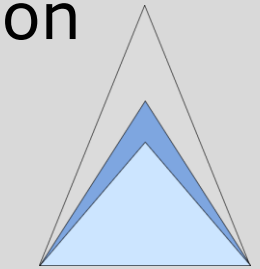
- Simple Testbench Visualization
  - Tree View
  - Diagram



Logging

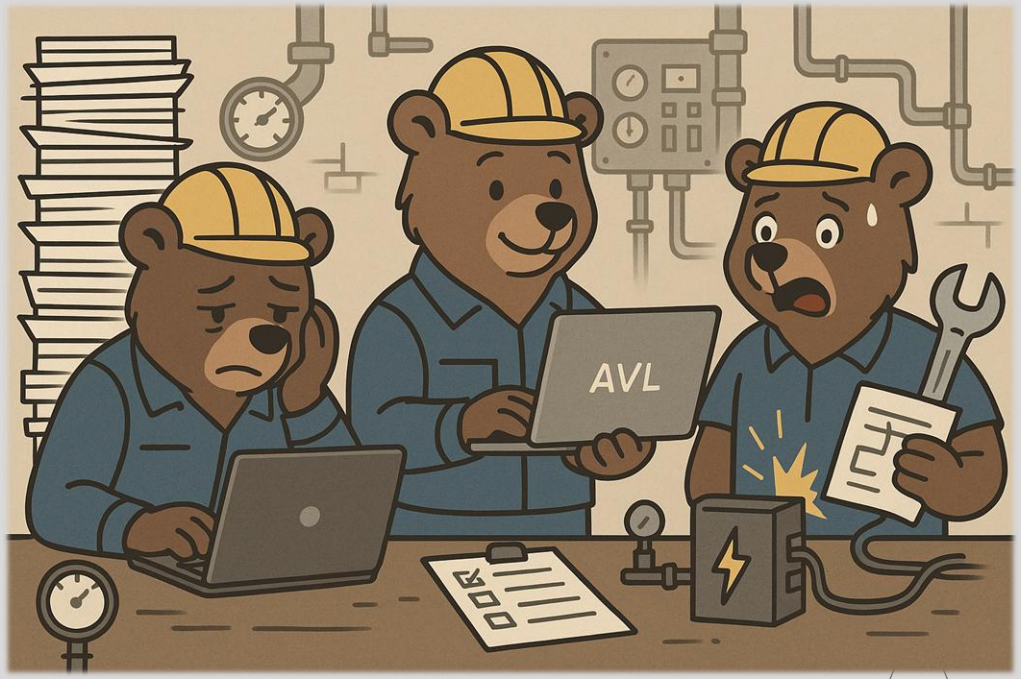
- AVL logging sits on-top of CocoTB logging
- Allows log messages to be exported into any Pandas supported file format
  - csv, json, yml, markdown, rst, text
- Human readable and machine readable
- Trivial to group, sort, filter and prioritize messages and errors using Pandas SQL style conditions
- All implemented in simple class
  - Messages can be modified / translated using common python libraries like Google Translate





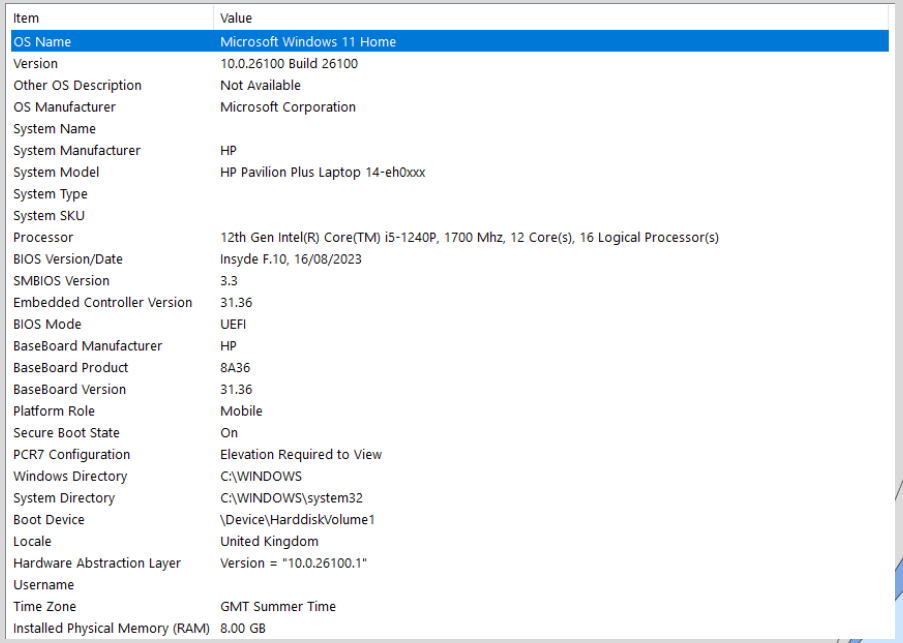
Conclusion

- AVL aims to be the Goldilocks verification library
- Extends CocoTB to unlock the full power of python as a verification language
- Simplifies the strong methodology of UVM allowing development of UVCs



Availability

- AVL 0.1.0 is available
  - https://github.com/projectapheleia/avl
  - https://avl-core.readthedocs.io/en/latest/index.html
  - https://pypi.org/project/avl-core/
  - pip install avl-core
- 100% license free
- Developed on mid-level PC



Item	Value
OS Name	Microsoft Windows 11 Home
Version	10.0.26100 Build 26100
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	
System Manufacturer	HP
System Model	HP Pavilion Plus Laptop 14-eh0xxx
System Type	
System SKU	
Processor	12th Gen Intel(R) Core(TM) i5-1240P, 1700 Mhz, 12 Core(s), 16 Logical Processor(s)
BIOS Version/Date	Insyde F.10, 16/08/2023
SMBIOS Version	3.3
Embedded Controller Version	31.36
BIOS Mode	UEFI
BaseBoard Manufacturer	HP
BaseBoard Product	8A36
BaseBoard Version	31.36
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United Kingdom
Hardware Abstraction Layer	Version = "10.0.26100.1"
Username	
Time Zone	GMT Summer Time
Installed Physical Memory (RAM)	8.00 GB

Questions?