2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Introduction to the Apheleia Verification Library

Andy Bond

Project Apheleia

accellera SYSTEMS INITIATIVE

Tutorial Structure

- Overview of AVL’s key concepts
- Setup and Prerequisites
- Fully worked examples
- Question and Answers

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Overview

accellera SYSTEMS INITIATIVE

Overview – What is AVL

- The Apheleia Verification Library (AVL) is an open-source python library
- 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

Overview – Who is AVL Aimed At?

- Novices and students
- Industry experts
- Hobbyists
- Professionals
- Anyone who wants to spend more time doing verification and less time developing code

Pros

- Successful at bringing standardization to the verification industry
- Strong methodology
- Wide range of available VIP
- Well understood terminology

Cons

- Limited to major EDA vendors with implementation specific version
- Inconsistent views on best practices
- Code intensive development

Pros

- 100% Open-Source
- Universal simulator support
- Active User Community
- Near Zero compile time
- Widely used an understood language
- Rich and diverse range of useful libraries

Cons

- No common methodology
- Limited available VIP
- Software centric
- Different

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
- Searchable Trace

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
	float	avl.Half / avl.Fp16
shortreal	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.Uint(250, width=8)
b = avl.Uint(10, width=8)

assert a + b == 4              # 250 + 10 = 260 % 256 = 4
c = avl.Uint(a, width=8)
c += b
assert c == 4                  # 250 + 10 = 260 % 256 = 4

assert a - b == 240            # 250 - 10 = 240
assert b - a == 16             # 10 - 250 = -240 % 256 = 16
c = avl.Uint(b, width=8)
c -= a
assert c == 16                 # 10 - 250 = -240 % 256 = 16

assert a * b == 196            # 250 * 10 = 2500 % 256 = 196
c = avl.Uint(a, width=8)
c *= b
assert c == 196                # 250 * 10 = 2500 % 256 = 196

assert a // b == 25            # 250 // 10 = 25
c = avl.Uint(a, width=8)
c //= b
assert c == 25                 # 250 // 10 = 25

assert a % b == 0              # 250 % 10 = 0
c = avl.Uint(a, width=8)
c %= b
assert c == 0                  # 250 % 10 = 0
```

AVL Variables

- Native floating-point values based on NumPy
- Helper functions to interact with hardware

```
a = avl.Fp16(2.0)
b = avl.Fp16(3.0)

# Arithmetic
assert a + b == 5.0           # 2.0 + 3.0 = 5.0
c = avl.Fp16(a)
c += b
assert c == 5.0               # 2.0 + 3.0 = 5.0

assert a - b == -1.0          # 2.0 - 3.0 = -1.0
c = avl.Fp16(a)
c -= b
assert c == -1.0              # 2.0 - 3.0 = -1.0

assert a * b == 6.0           # 2.0 * 3.0 = 6.0
c = avl.Fp16(a)
c *= b
assert c == 6.0               # 2.0 * 3.0 = 6.0

assert b / a == 1.5           # 3.0 / 2.0 = 1.5
c = avl.Fp16(b)
c /= a
assert c == 1.5               # 3.0 / 2.0 = 1.5

# Power
assert a ** 2 == 4.0          # 2.0 ** 2 = 4.0
c = avl.Fp16(a)
c **= 2
assert c == 4.0               # 2.0 ** 2 = 4.0
```

AVL Variables

- Structures
  - Verilator and some other simulators flattens structs
  - Helper class provided to be simulator agnostic

```
class packed_struct_t(avl.Struct):
    single_bit : avl.Bool = avl.Bool(False)
    multi_bit : avl.Uint32 = avl.Uint32(0)
    state_enum : avl.Enum = avl.Enum("S0", {"S0" : 0, "S1" : 1, "S2" : 2})
```

Constrained Random

- UVM benefits from the constrained random features of SystemVerilog
- Python random supports randomization, but lacks constraints
- There are many theorem solvers available in Python
  - Each has benefits and limitations
  - Can be confusing to decide on best approach

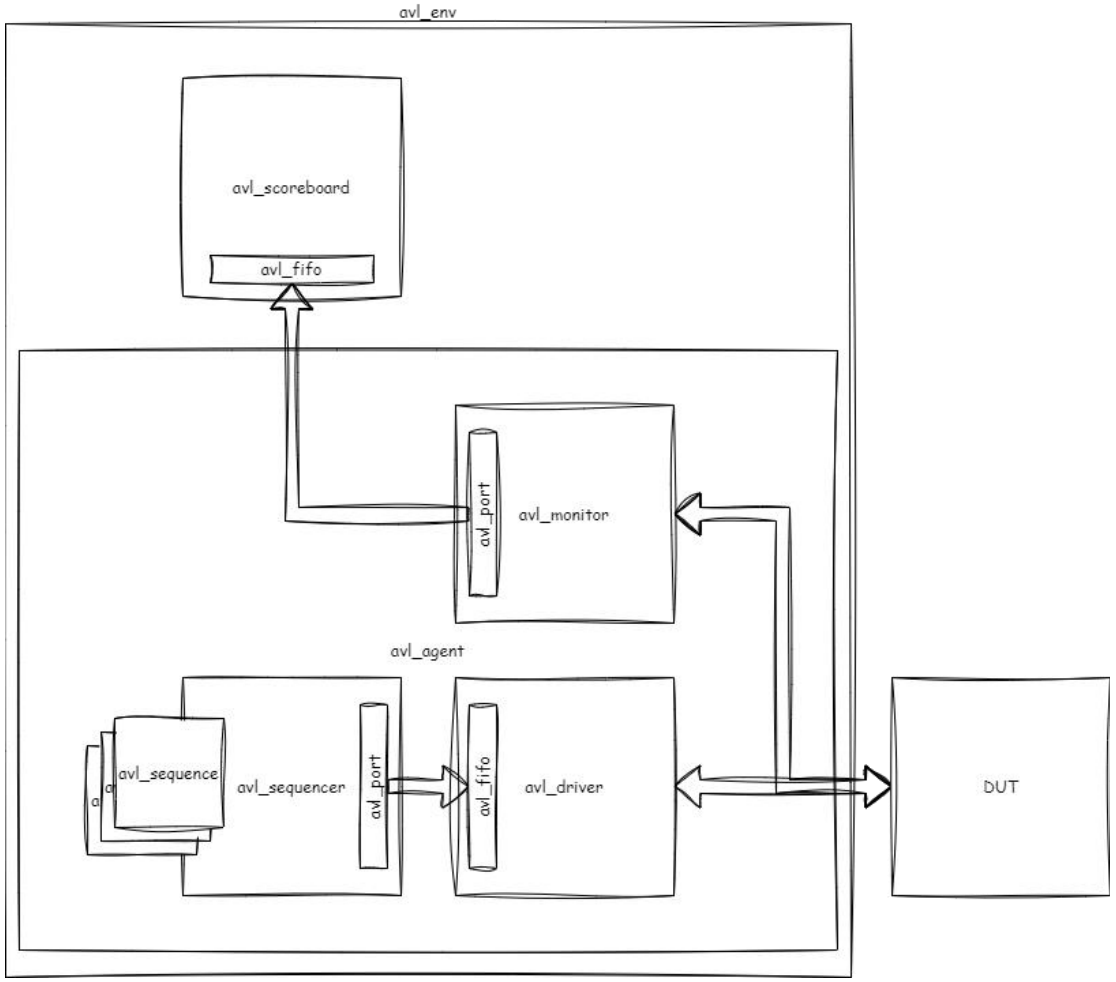
Constrained Random

- AVL utilizes Z3 – an open-source theorem prover from Microsoft
- Supports bool, int, uint, enum and **float** numbers and wide variables
- Supports hard and soft constraints
- Well documented and well maintained

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

AVL Methodology

- AVL follows the UVM methodology
- Familiar and consistent
- Terminology and behaviour maintained where appropriate
- No need for parameterization
- Direct access
  - No requirement for virtual interfaces



Factory

- UVM factory was a fudge due to language limitation
- AVL factory built in natively
- User extendable specificity function to decide override precidence

```
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)
```

Phases

- Phases are useful, but the need for them varies based on the type of testbench
- AVL supports adding, inserting and removing phase, but by default only provides 2 – Run & Report

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Setup and Prerequisites

accellera SYSTEMS INITIATIVE

Setup and Prerequisites

- AVL is designed to be fully Open-Source
- AVL can be run on any 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

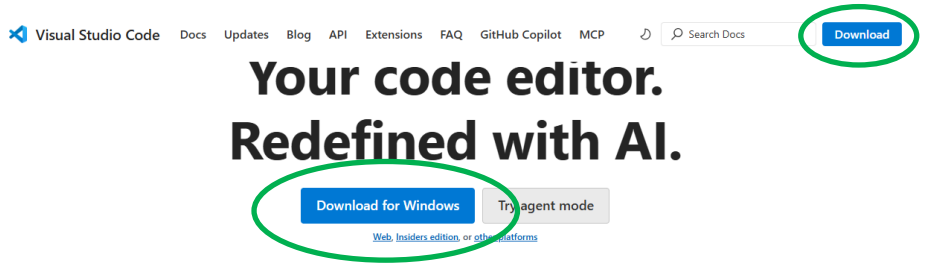
Setup

- This tutorial has been developed on Ubuntu running on a windows laptop under [WSL]

PowerShell Copy

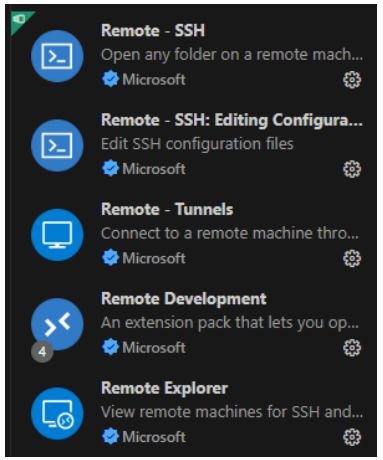
```
wsl --install
```

- The demos are run under [VSCode]

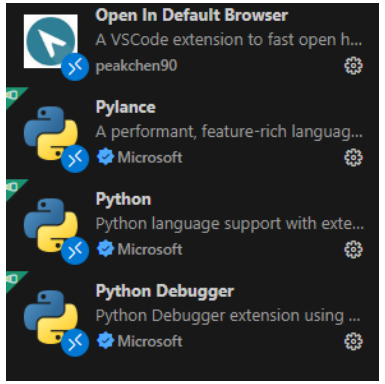


Setup

- Using VSCode with WSL
  - https://code.visualstudio.com/docs/remote/wsl



- For a better editor experience Python, SystemVerilog and Browser extensions are added to VSCode



Setup

- All examples are run using Verilator

Installation

This section discusses how to install Verilator.

Package Manager Quick Install

Using a distribution's package manager is the easiest way to get started. (Note packages are unlikely to have the most recent version, so Git Quick Install might be a better alternative.) To install as a package:

```
apt-get install verilator   # On Ubuntu
```

Setup

- Waveforms can be viewed using GTKWave

GTKWave

GTKWave is a fully featured GTK+ based wave viewer for Unix and Win32 which reads FST, and GHW files as well as standard Verilog VCD/EVCD files and allows their viewing.

Building GTKWave from source

Installing dependencies

Debian, Ubuntu:

```
apt install build-essential meson gperf flex desktop-file-utils libgtk-3-dev \
           libbz2-dev libjudy-dev libgirepository1.0-dev
```

Fedora:

```
dnf install meson gperf flex glib2-devel gcc gcc-c++ gtk3-devel \
           gobject-introspection-devel desktop-file-utils tcl
```

macOS:

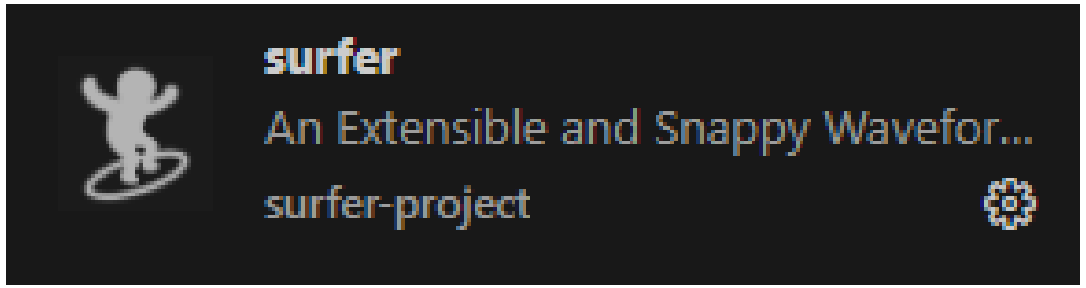
```
brew install desktop-file-utils shared-mime-info       \
             gobject-introspection gtk-mac-integration \
             meson ninja pkg-config gtk+3 gtk4
```

Building GTKWave

```
git clone "https://github.com/gtkwave/gtkwave.git"
cd gtkwave
meson setup build && cd build && meson install
```

Setup

- Surfer is a preferred Waveform viewer with VSCode integration



Setup

- All material for this tutorial is on GitHub
  - https://github.com/projectapheleia/dvcon_europe25
- To create a virtual environment and install avl-core
  - source avl.sh
- Full AVL documentation is available on ReadTheDocs
  - https://avl-core.readthedocs.io/en/latest/index.html

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

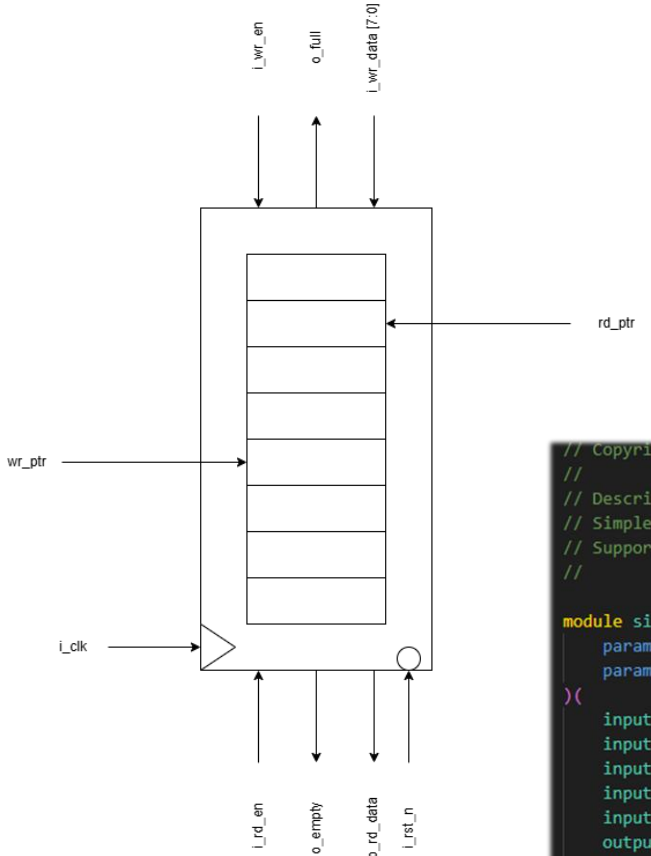
Tutorial 1

Simple FIFO

accellera SYSTEMS INITIATIVE

Design Under Test

- The DUT is a simple FIFO
- 100MHz Clock
- Async – active low reset
- Width : 8 bits
- Depth : 8 entries
- Full / Empty indicators




```
// Copyright 2025 ProjectApheleia
//
// Description:
// Simple fixed width FIFO
// Supports bypass and read/write on full
//

module simple_fifo #(
    parameter WIDTH = 8,       // Width of each data entry
    parameter DEPTH = 8        // Depth of the FIFO (number of entries)
)(
    input  logic             i_clk,       // Clock
    input  logic             i_rst_n,     // Active-low reset
    input  logic             i_wr_en,     // Write enable
    input  logic             i_rd_en,     // Read enable
    input  logic [WIDTH-1:0] i_wr_data,   // Write data
    output logic [WIDTH-1:0] o_rd_data,   // Read data
    output logic             o_full,      // FIFO full flag
    output logic             o_empty      // FIFO empty flag
);
```

Phase 0

1. Extend the environment to drive the clock and reset signals
2. Add a timeout (1ms)
3. Confirm FIFO is empty after reset

```python
import avl
import cocotb

class simple_fifo_env(avl.Env):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.info("Creating simple_fifo_env:...")

@cocotb.test
async def test(dut):
    env = simple_fifo_env("env", None)
```

https://avl-core.readthedocs.io/en/latest/methodology/env.html

Phase 0

- Register handle to DUT

```python
@cocotb.test
async def test(dut):
    # Register the hdl with the factory for easy access
    avl.Factory.set_variable("*.hdl", dut)

    env = simple_fifo_env("env", None)
    await env.start()
```

```python
import avl
import cocotb
from cocotb.triggers import Timer, RisingEdge

class simple_fifo_env(avl.Env):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.info("Creating simple_fifo_env:...")

        # Register the HDL module
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)
```

Phase 0

- Register handle to DUT
- Create a run_phase
- Create a clock

```python
import avl
import cocotb
from cocotb.triggers import Timer, RisingEdge

class simple_fifo_env(avl.Env):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.info("Creating simple_fifo_env:...")

        # Register the HDL module
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

    async def run_phase(self):
        self.raise_objection()

        # Create a clock
        cocotb.start_soon(self.clock(self.hdl.i_clk, freq_mHz=100))
```

Phase 0

- Register handle to DUT
- Create a run_phase
- Create a clock
- Create a reset
- Create a timeout

```python
import avl
import cocotb
from cocotb.triggers import Timer, RisingEdge

class simple_fifo_env(avl.Env):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.info("Creating simple_fifo_env:...")

        # Register the HDL module
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

    async def run_phase(self):
        self.raise_objection()

        # Create a clock
        cocotb.start_soon(self.clock(self.hdl.i_clk, freq_mHz=100))

        # Create a reset
        cocotb.start_soon(self.async_reset(self.hdl.i_rst_n, duration=100, units="ns", active_high=False))

        # Create a Timeout
        cocotb.start_soon(self.timeout(duration=1, units="ms"))
```

Phase 0

- Register handle to DUT
- Create a run_phase
- Create a clock
- Create a reset
- Create a timeout
- Add the reset check

```python
import avl
import cocotb
from cocotb.triggers import Timer, RisingEdge

class simple_fifo_env(avl.Env):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.info("Creating simple_fifo_env:...")

        # Register the HDL module
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

    async def run_phase(self):
        self.raise_objection()

        # Create a clock
        cocotb.start_soon(self.clock(self.hdl.i_clk, freq_mHz=100))

        # Create a reset
        cocotb.start_soon(self.async_reset(self.hdl.i_rst_n, duration=100, units="ns", active_high=False))

        # Create a Timeout
        cocotb.start_soon(self.timeout(duration=1, units="ms"))

        # Check reset
        while True:
            await RisingEdge(self.hdl.i_clk)
            if self.hdl.i_rst_n.value == 1:
                break
        assert self.hdl.o_full.value == 0, "FIFO should be empty after reset"
        assert self.hdl.o_empty.value == 1, "FIFO should be empty after reset"

        # Run for a while
        await Timer(2, unit="us")

        self.drop_objection()
```

Phase 0

- Run the test
  - make sim

```
COCOTB_TEST_MODULES=simple_fifo_tb COCOTB_TESTCASE= COCOTB_TEST_FILTER= COCOTB_TOPLEVEL=simple_fifo TOPLEVEL_LANG=verilog \
         sim_build/Vtop  --trace --trace-structs
     -.--ns INFO     gpi                        ..mbed/gpi_embed.cpp:93   in _embed_init_python            Using Python 3.12.4 interpreter at /home/abond/projects/TEMP/dvcon_europe25/venv/bin/python3
     -.--ns INFO     gpi                        ../gpi/GpiCommon.cpp:79   in gpi_print_registered_impl     VPI registered
     0.00ns INFO     cocotb                     Running on Verilator version 5.040 2025-08-30
     0.00ns INFO     cocotb                     Seeding Python random module with 1759574287
     0.00ns INFO     cocotb                     Initialized cocotb v2.0.0 from /home/abond/projects/TEMP/dvcon_europe25/venv/lib/python3.12/site-packages/cocotb
     0.00ns INFO     cocotb.regression          pytest not found, install it to enable better AssertionError messages
     0.00ns INFO     cocotb                     Running tests
     0.00ns INFO     cocotb.regression          running simple_fifo_tb.test (1/1)
     0.00ns INFO     cocotb                     Creating simple_fifo_env:...
   100.00ns WARNING  cocotb.Test test.test      FIFO should be empty after reset
                                                Traceback (most recent call last):
                                                  File "/home/abond/projects/TEMP/dvcon_europe25/tutorials/t1_simple_fifo/phase0_solution/cocotb/simple_fifo_tb.py", line 49, in test
                                                    await env.start()
                                                  File "/home/abond/projects/TEMP/dvcon_europe25/venv/lib/python3.12/site-packages/avl/_core/component.py", line 131, in start
                                                    await self._hierarchical_func_(fn_name)
                                                  File "/home/abond/projects/TEMP/dvcon_europe25/venv/lib/python3.12/site-packages/avl/_core/component.py", line 81, in _hierarchical_func_
                                                    await fn(*args, **kwargs)
                                                  File "/home/abond/projects/TEMP/dvcon_europe25/tutorials/t1_simple_fifo/phase0_solution/cocotb/simple_fifo_tb.py", line 35, in run_phase
                                                    assert self.dut.o_full.value == 0, "FIFO should be empty after reset"
                                                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
                                                AssertionError: FIFO should be empty after reset
   100.00ns WARNING  cocotb.regression          simple_fifo_tb.test failed
   100.00ns INFO     cocotb.regression          ********************************************************************************
                                                ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                ********************************************************************************
                                                ** simple_fifo_tb.test            FAIL         100.00          0.00      56833.39  **
                                                ********************************************************************************
                                                ** TESTS=1 PASS=0 FAIL=1 SKIP=0               100.00          0.00      40896.10  **
                                                ********************************************************************************
```

- Error on o_full due to width of variable

```
localparam ADDR_WIDTH = $clog2(DEPTH);  // Address width for FIFO depth
typedef logic [ADDR_WIDTH-1:0] ptr_t;   // Pointer type
typedef logic [WIDTH-1:0]      data_t;  // Data type
typedef logic [ADDR_WIDTH-1:0] count_t; // Counter type
```

```
localparam ADDR_WIDTH = $clog2(DEPTH);  // Address width for FIFO depth
typedef logic [ADDR_WIDTH-1:0] ptr_t;   // Pointer type
typedef logic [WIDTH-1:0]      data_t;  // Data type
typedef logic [ADDR_WIDTH:0]   count_t; // Counter type
```

Phase 1

- Create
  - Agent
  - Driver
  - Sequencer
- Create a random sequence
- Create a monitor with built in checks

Phase 1

- Create agent and driver

```python
class simple_fifo_agent(avl.Agent):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

        # Create a driver
        self.driver = simple_fifo_driver("driver", self)

        # Create a monitor
        self.monitor = simple_fifo_monitor("monitor", self)

        # Create a sequencer
        self.sequencer = avl.Sequencer("sequencer", self) # No need to override

        # Create a sequence
        self.sequence = simple_fifo_sequence("sequence", self)

        # Connect the driver to the sequencer
        self.sequencer.seq_item_export.connect(self.driver.seq_item_port)

        # Assign sequencer to the sequencer
        self.sequence.set_sequencer(self.sequencer)

    async def run_phase(self):
        await self.sequence.start()
```

```python
class simple_fifo_driver(avl.Driver):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

    async def reset(self):
        self.hdl.i_wr_en.value = 0
        self.hdl.i_wr_data.value = 0
        self.hdl.i_rd_en.value = 0

    async def pop(self):
        while True:
            self.hdl.i_rd_en.value = 1
            await RisingEdge(self.hdl.i_clk)

    async def push(self, item):
        self.hdl.i_wr_en.value = 1
        self.hdl.i_wr_data.value = item.value.value
        while True:
            if not bool(self.hdl.o_full.value):
                break
        item.set_event("done")
        await RisingEdge(self.hdl.i_clk)
        self.hdl.i_wr_en.value = 0
        self.hdl.i_wr_data.value = 0

    async def run_phase(self):
        await self.reset()

        # Start pops in background
        cocotb.start_soon(self.pop())

        while True:
            item = await self.seq_item_port.blocking_get()
            while True:
                await RisingEdge(self.hdl.i_clk)
                if not bool(self.hdl.i_rst_n.value):
                    await self.reset()
                else:
                    break
            cocotb.start_soon(self.push(item))
```

Phase 1

- Create
  - Agent
  - Driver
  - Sequencer
- Create a random sequence

```
class simple_fifo_item(avl.SequenceItem):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.value = avl.Logic(0, width=8, fmt=hex)

class simple_fifo_sequence(avl.Sequence):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.n_items = avl.Factory.get_variable(f"{self.get_full_name()}.n_items", 4)

    async def body(self):
        sqr = self.get_sequencer()

        sqr.raise_objection()
        for _ in range(self.n_items):
            item = simple_fifo_item("item", self)
            await self.start_item(item)
            item.randomize()
            await self.finish_item(item)
        sqr.drop_objection()
```

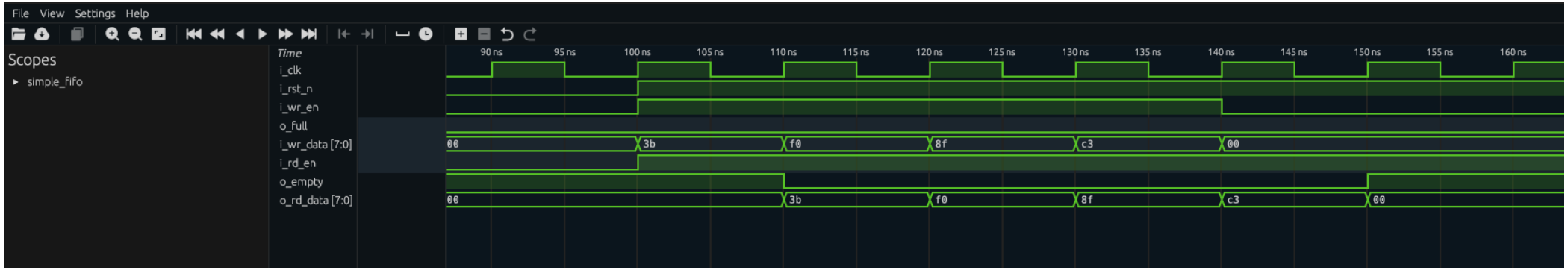
Phase 1

- Create
  - Agent
  - Driver
  - Sequencer
- Create a random sequence
- Create a monitor with built in checks

```python
class simple_fifo_monitor(avl.Monitor):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)

        # Build checks into monitor as so simple
        self.fifo = avl.Fifo(8)

    async def run_phase(self):
        while True:
            await RisingEdge(self.hdl.i_clk)

            if not bool(self.hdl.i_rst_n.value):
                self.fifo.clear()
                continue

            assert self.hdl.o_full.value == (len(self.fifo) == 8), "FIFO full signal mismatch"
            assert self.hdl.o_empty.value == (len(self.fifo) == 0), "FIFO empty signal mismatch"

            if bool(self.hdl.i_rd_en.value) and not bool(self.hdl.o_empty.value):
                assert self.fifo.pop(0) == self.hdl.o_rd_data.value

            if bool(self.hdl.i_wr_en.value) and not bool(self.hdl.o_full.value):
                self.fifo.append(self.hdl.i_wr_data.value)
```

Phase 1

- make sim



- FIFO never fills as we pop immediately

Phase 2

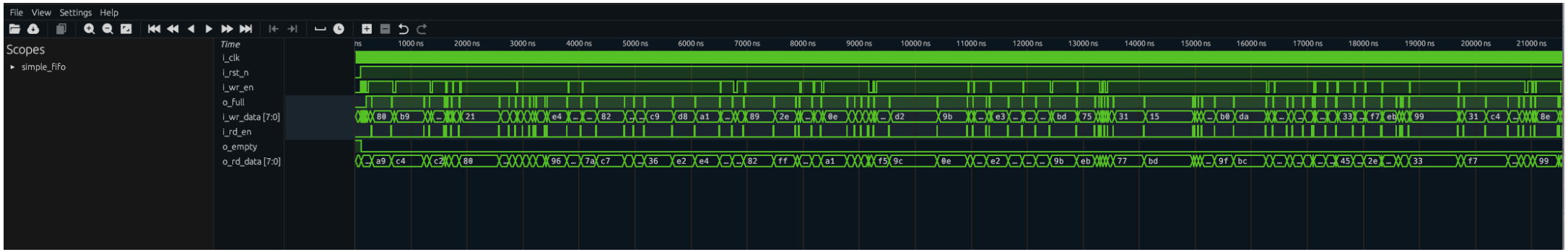
- Add rate limiters to driver
- Increase number of items

Phase 2

- Add rate limiters to driver

```python
class simple_fifo_driver(avl.Driver):
    def __init__(self, name, parent):
        super().__init__(name, parent)
        self.hdl = avl.Factory.get_variable(f"{self.get_full_name()}.hdl", None)
        self.wr_rate = avl.Factory.get_variable(f"{self.get_full_name()}.wr_rate", 100)
        self.rd_rate = avl.Factory.get_variable(f"{self.get_full_name()}.rd_rate", 100)

    async def reset(self):
        self.hdl.i_wr_en.value = 0
        self.hdl.i_wr_data.value = 0
        self.hdl.i_rd_en.value = 0

    async def pop(self):
        while True:
            self.hdl.i_rd_en.value = 0
            while random.randint(0, 100) > self.rd_rate:
                await RisingEdge(self.hdl.i_clk)
            self.hdl.i_rd_en.value = 1
            await RisingEdge(self.hdl.i_clk)

    async def push(self, item):
        while random.randint(0, 100) > self.wr_rate:
            await RisingEdge(self.hdl.i_clk)

        self.hdl.i_wr_en.value = 1
        self.hdl.i_wr_data.value = item.value.value
        while True:
            if bool(self.hdl.o_full.value):
                await RisingEdge(self.hdl.i_clk)
            else:
                break
        item.set_event("done")
        await RisingEdge(self.hdl.i_clk)
        self.hdl.i_wr_en.value = 0
        self.hdl.i_wr_data.value = 0
```

Phase 2


File View Settings Help
Scopes
simple_fifo
Time
i_clk
i_rst_n
i_wr_en
o_full
i_wr_data [7:0]
i_rd_en
o_empty
o_rd_data [7:0]
1000 ns
2000 ns
3000 ns
4000 ns
5000 ns
6000 ns
7000 ns
8000 ns
9000 ns
10000 ns
11000 ns
12000 ns
13000 ns
14000 ns
15000 ns
16000 ns
17000 ns
18000 ns
19000 ns
20000 ns
21000 ns

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Tutorial 2

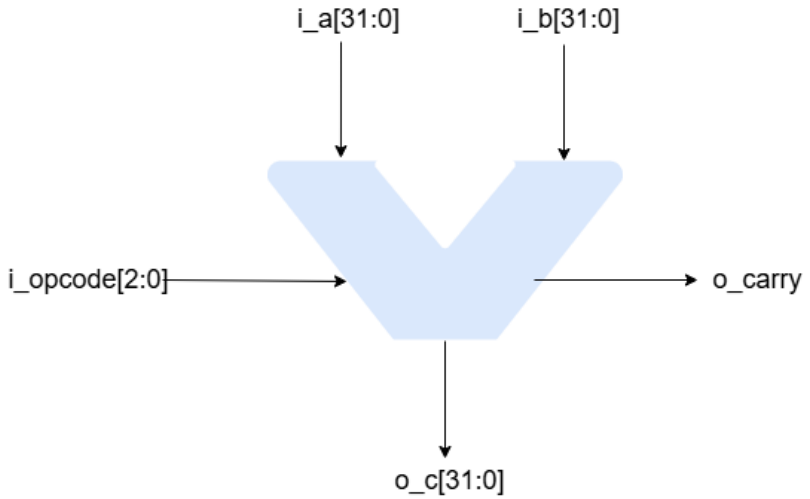
ALU

accellera SYSTEMS INITIATIVE

Design Under Test

- Combinatorial ALU unit

Opcode	Description
3’b000	NOP
3’b001	ADD
3’b010	SUB
3’b011	AND
3’b100	OR
3’b101	XOR
3’b110	COMP
3’b111	ILLEGAL



Phase 0

- Using a template create testbench with
  - Sequence / Sequence Item
  - Sequencer
  - Driver
  - Model
  - Scoreboard
- Visualize
  - Tree
  - Diagram

Phase 0

- Using a template create testbench with
  - Sequence / Sequence Item

```
class alu_item(avl.SequenceItem):
    def __init__(self, name, parent_sequence):
        super().__init__(name, parent_sequence)
        self.set_field_attributes("name", compare=False)

        self.opcode = avl.Enum("NOP", {"NOP" : 0, "ADD" : 1, "SUB" : 2, "AND" : 3, "OR" : 4, "XOR" : 5, "COMP" : 6, "ILLEGAL" : 7})
        self.a      = avl.Uint32(0, fmt=hex)
        self.b      = avl.Uint32(0, fmt=hex)
        self.c      = avl.Uint32(0, fmt=hex)
        self.carry  = avl.Bool(False)
```

No need for sequence or sequencer – built into template

Phase 0

- Using a template create testbench with
  - Driver

```python
class alu_driver(avl.templates.VanillaDriver):
    async def clear(self):
        self.hdl.i_a.value = 0
        self.hdl.i_b.value = 0
        self.hdl.i_opcode.value = 0

    async def reset(self):
        await self.clear()
        while True:
            await FallingEdge(self.rst)
            await self.clear()

    async def run_phase(self):
        cocotb.start_soon(self.reset())

        while True:
            item = await self.seq_item_port.blocking_get()
            while True:
                await RisingEdge(self.clk)
                if bool(self.rst.value):
                    break

            self.hdl.i_opcode.value = int(item.opcode)
            self.hdl.i_a.value = int(item.a)
            self.hdl.i_b.value = int(item.b)
            item.set_event("done")

    async def report_phase(self):
        self.raise_objection()
        for _ in range(10):
            await RisingEdge(self.clk)
            await self.clear()
        self.drop_objection()
```

Phase 0

- Using a template create testbench with
  - Model
  - Scoreboard

```
class alu_model(avl.templates.VanillaModel):

    async def run_phase(self):
        self.info("Running ALU model...")
        while True:
            orig = await self.item_port.blocking_get()
            item = alu_item("model_item", None)
            item.opcode.value  = orig.opcode.value
            item.a.value = orig.a.value
            item.b.value = orig.b.value

            if item.opcode == item.opcode.NOP:
                c = avl.Logic(0, fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.ADD:
                c = avl.Logic((item.a.value + item.b.value), fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.SUB:
                c = avl.Logic((item.a.value - item.b.value), fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.AND:
                c = avl.Logic((item.a.value & item.b.value), fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.OR:
                c = avl.Logic((item.a.value | item.b.value), fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.XOR:
                c = avl.Logic((item.a.value ^ item.b.value), fmt=hex, auto_random=False, width=33)
            elif item.opcode == item.opcode.COMP:
                c = avl.Logic((item.a.value == item.b.value), fmt=hex, auto_random=False, width=33)
            else:
                raise ValueError("Illegal opcode")

            item.c     = avl.Uint32(c)
            item.carry = avl.Bool((c >> 32))
            self.item_export.write(item)
```

No need for scoreboard– built into template

Phase 0

- make run
  - $error – opcode[7] is illegal

```
ODULE=simple_alu_tb TESTCASE= TOPLEVEL= TOPLEVEL_LANG=verilog \
        sim_build/Vtop  --trace --trace-structs
     -.--ns INFO     gpi                                ..mbed/gpi_embed.cpp:108  in set_program_name_in_venv       Using Python virtual environment interpreter at /home/abond/projects/dvcon_europe25/venv/bin/python
     -.--ns INFO     gpi                                ../gpi/GpiCommon.cpp:101  in gpi_print_registered_impl      VPI registered
    0.00ns INFO     cocotb                             Running on Verilator version 5.020 2024-01-01
    0.00ns INFO     cocotb                             Running tests with cocotb v1.9.2 from /home/abond/projects/dvcon_europe25/venv/lib/python3.12/site-packages/cocotb
    0.00ns INFO     cocotb                             Seeding Python random module with 1750609803
    0.00ns INFO     cocotb.regression                  Found test simple_alu_tb.test
    0.00ns INFO     cocotb.regression                  running test (1/1)
[120000] %Error: simple_alu.sv:55: Assertion failed in simple_alu: Invalid opcode: 111
%Error: rtl/simple_alu.sv:55: Verilog $stop
borting...
```

Phase 0

```python
class alu_item(avl.SequenceItem):
    def __init__(self, name, parent_sequence):
        super().__init__(name, parent_sequence)
        self.set_field_attributes("name", compare=False)

        self.opcode = avl.Enum("NOP", {"NOP" : 0, "ADD" : 1, "SUB" : 2, "AND" : 3, "OR" : 4, "XOR" : 5, "COMP" : 6, "ILLEGAL" : 7})
        self.a      = avl.Uint32(0, fmt=hex)
        self.b      = avl.Uint32(0, fmt=hex)
        self.c      = avl.Uint32(0, fmt=hex)
        self.carry  = avl.Bool(False)

        # Add constraints
        self.add_constraint("c_no_illegal", lambda x : x != self.opcode.ILLEGAL, self.opcode)
```

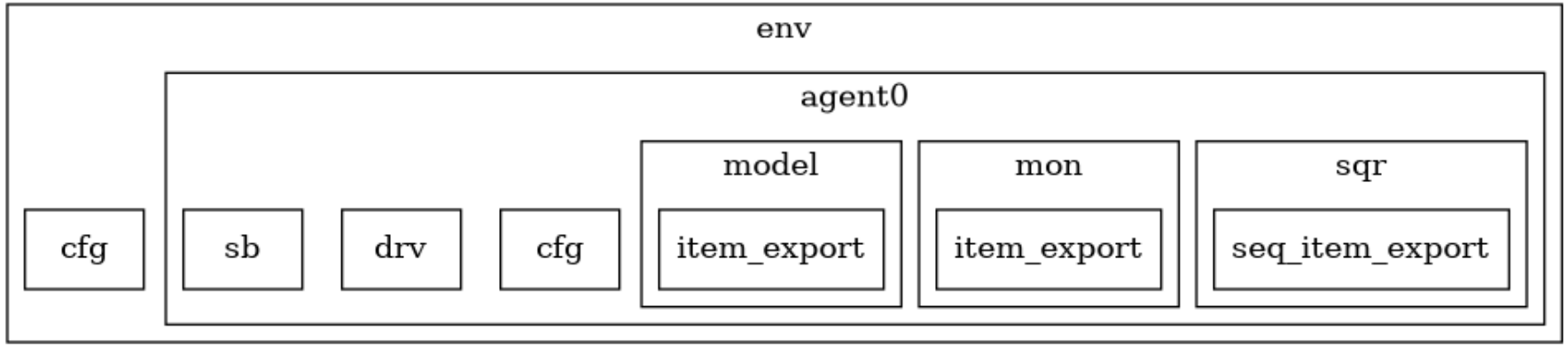
Add constraint to avoid illegal instructions

Phase 0

- Visualize
  - True
  - Diagram

```
e = avl.templates.VanillaEnv("env", None)
print(avl.Visualization.tree(e))
avl.Visualization.diagram(e)
```

```
     0.00ns INFO     cocotb.regression                  Running test (1/1)
env
├─ cfg
└─ agent0
   ├─ cfg
   ├─ sqr
   │  └─ seq_item_export
   ├─ drv
   ├─ mon
   │  └─ item_export
   ├─ model
   │  └─ item_export
   └─ sb
```



Phase 1

- Add constraints to a and b to generate more interesting values

Phase 1

- Add constraints to a and b to generate more interesting values

```
# Add some interesting values to a and b
interesting_values = [self.a.get_min(), self.a.get_min()+1, self.a.get_max()-1, self.a.get_max()]
interesting_values.extend([1 << i for i in range(32)])
interesting_values.extend([~(1 << i) for i in range(32)])
weights = [1] * len(interesting_values)
self.add_constraint("c_a", lambda x,y : x == random.choices(interesting_values + [y],
                                                            weights=weights + [100])[0],
                                                            self.a, random.randint(self.a.get_min(), self.a.get_max()))
self.add_constraint("c_b", lambda x,y : x == random.choices(interesting_values + [y],
                                                            weights=weights + [100])[0], self.b,
                                                            random.randint(self.b.get_min(), self.b.get_max()))
```

Phase 2

- Add functional coverage
- Generate coverage report

Phase 2

- Add functional coverage

```
def __init__(self, name, parent_env):
    super().__init__(name, parent_env)

    # Coverage
    self.item = alu_item("coverage_item", None)

    self.cg        = avl.Covergroup("alu_cg", self)

    # Cover all opcodes
    self.cp_opcode = self.cg.add_coverpoint("opcode", lambda: self.item.opcode)
    for k,v in self.item.opcode.values.items():
        self.cp_opcode.add_bin(k, v, illegal=(k=="ILLEGAL"))

    # Cover all interesting values for a and b
    self.cp_a = self.cg.add_coverpoint("a", lambda: self.item.a)
    self.cp_b = self.cg.add_coverpoint("b", lambda: self.item.b)
    for v in alu_item.interesting_values:
        self.cp_a.add_bin(f'{v}', v)
        self.cp_b.add_bin(f'{v}', v)

    # Cover the carry result
    self.cp_carry = self.cg.add_coverpoint("carry", lambda: self.item.carry)
    for t in (True, False):
        self.cp_carry.add_bin(f'{t}', t)

    # Cross coverage carry X opcode
    self.cc = self.cg.add_covercross("carry_X_opcode", self.cp_carry, self.cp_opcode)
    # Remove opcodes which can't generate a carry
    rbins = []
    for k,v in self.cc._bins_.items():
        if k.startswith("True") and not (k.endswith("ADD") or k.endswith("SUB")):
            rbins.append(k)
    for k in rbins:
        self.cc.remove_bin(k)

async def report_phase(self):
    print(self.cg.report(full=False))
```

Phase 2

- Add functional coverage
- Generate coverage report
  - avl-coverage-analysis --path .

Coverage Report

Show 10 entries Search:

name	total bins	covered bins	coverage
Filter name	Filter total bins	Filter covered bins	Filter coverage
coverage	146.0	42.0	28.767123

Showing 1 to 1 of 1 entries Previous 1 Next

Verification Components

- Some AMBA protocol components are now available
  - APB
  - AXI-STREAM
  - AXI

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Thank You

https://github.com/projectapheleia/avl

https://avl-core.readthedocs.io/en/latest/index.html

accellera SYSTEMS INITIATIVE

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Questions And Answers

accellera SYSTEMS INITIATIVE