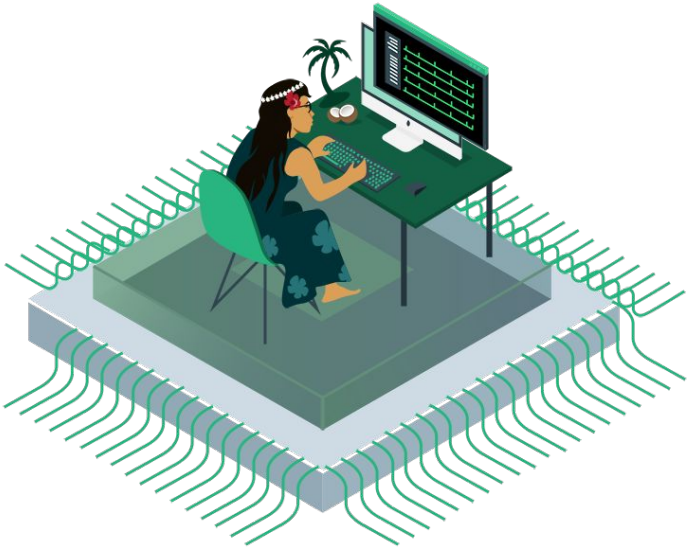
cocotb 2.0: Modernize your testbenches for even more productivity



Philipp Wagner
Kaleb Barrett

DVClub Europe
Oct 8, 2024

About Philipp

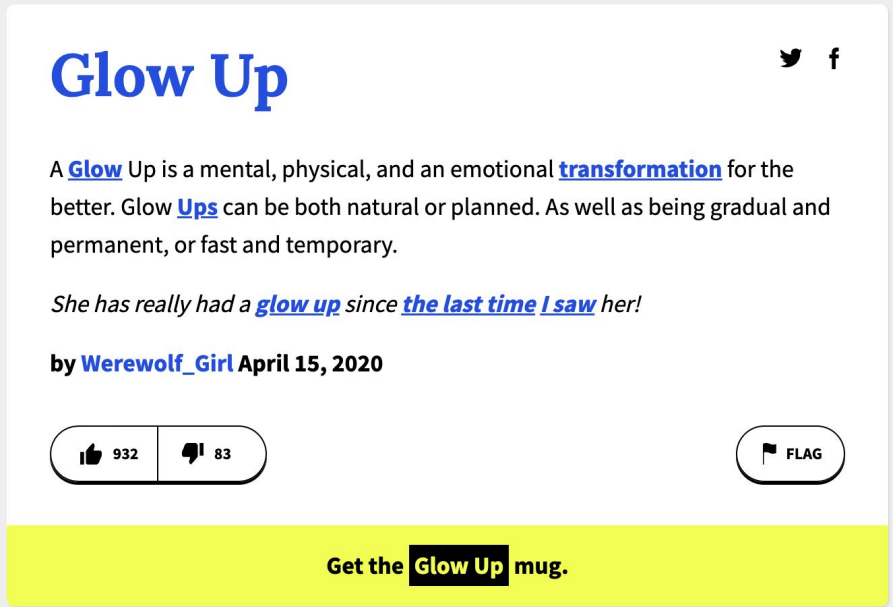
open source initiative

The FOSSi Foundation.

phw@ibm.com
philipp@fossi-foundation.org
@imphil on GitHub
@MrImphil@mastodon.social

Why cocotb 2.0?

- Needed API breaking changes
  - Replacing footguns
  - Refining public API
  - General code cleanup
- Break as little backwards compatibility as possible
  - ***NOT*** like Python 2 to Python 3



Project Automation

Python Runner

What is it?

- Designed to build most HDL designs
- Runs cocotb simulations
- Replacement for Makefiles
- Based on `cocotb-test`

Why prefer Python?

- Python > Make language
- Better cross-platform support
- pytest ecosystem

```python

def test_alu():
    sim = runner.Icarus()
    sim.build(
        sources=[
            "src/adder.sv",
            "src/multiplier.sv",
            "src/alu.sv",
        ],
        includes=[
            "src/"
        ],
        defines={
            "SIM": "1",
        },
    )
    sim.test(
        test_module="alu_tests",
        waves=True,
    )

```

Python Runner

Python Runner

```makefile

SIM := icarus

VERILOG_SOURCES := \
    src/adder.sv \
    src/multiplier.sv \
    src/alu.sv
COMPILE_ARGS += -DSIM=1 # turn on simulation-only code
COMPILE_ARGS += -Isrc/

MODULE := alu_tests
WAVES ?= 1

include $(shell cocotb-config --makefiles)/Makefile.sim

```

Makefile

```python

def test_alu():
    sim = runner.Icarus()
    sim.build(
        sources=[
            "src/adder.sv",
            "src/multiplier.sv",
            "src/alu.sv",
        ],
        includes=[
            "src/"
        ],
        defines={
            "SIM": "1",
        },
    )
    sim.test(
        test_module="alu_tests",
        waves=True,
    )

```

Python Runner

Python Runner

```python

@pytest.mark.parametrize("depth", [16, 64, 1024, 4096])
@pytest.mark.parametrize("width", [1, 8, 64, 5121])
@pytest.mark.parametrize("valid_fraction", [0.5, 1.0])
@pytest.mark.parametrize("ready_fraction", [0.5, 1.0])
def test_fifo(depth: int, width: int, valid_fraction: float, ready_fraction: float) -> None:

    sim = Questa()
    sim.build(
        build_dir=f"build_{depth}_{width}_{valid_fraction}_{ready_fraction}",
        parameters={
            "DEPTH": depth,
            "WIDTH": width,
        },
        sources=["fifo.vhd", "fifo_top.sv"], # mixed language build
        args=[VHDL("-2008")], # VHDL-only build args
    )
    sim.test(
        test_module="fifo_tests",
        plusargs=[
            f"+valid_fraction={valid_fraction}", f"+ready_fraction={ready_fraction}"
        ]
    )

```

cocotb

Test Discovery and Generation

Test Discovery Changes

What changed?

- `cocotb.test` decorator no longer returns Test objects
- Tests are placed in special attribute in module where they are defined

Why?

- Allows reusing test definitions
- Import test into different modules without adding the test to the module

Potential Incompatibilities

- Custom test generation utilities

```python

@pyuvm.test()
class TestA(uvm_test):
    def build_phase(self):
        ...

    async def run_phase(self):
        ...


@pyuvm.test()
class TestB(TestA):
    async def run_phase(self):
        ...  # different test, same structure

```

cocotb.parametrize()

What is it?

- Write test once, generate many related tests
- Replacement for `TestFactory`
- Inspired by `pytest.mark.parametrize`

Why replace `TestFactory`?

- Lists options near test function parameters
- Generates better tests names
- Familiar to pytest users

```

@cocotb.test
@cocotb.parametrize(
    valid_fraction=[0.5, 1.0],
    ready_fraction=[0.5, 1.0],
)
async def test_fifo(
    dut,
    valid_fraction: float,
    ready_fraction: float,
) -> None:
    ...

```

cocotb.parametrize()

```python

@cocotb.test
async def test_fifo(
    dut,
    valid_fraction: float,
    ready_fraction: float
) -> None:
    ...

tf = TestFactory(test_fifo)
tf.add_option("valid_fraction", [0.5, 1.0])
tf.add_option("ready_fraction", [0.5, 1.0])
tf.generate_tests()

```

```
**************************************************************************************
** TEST                                STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
**************************************************************************************
** example_tests.test_fifo_001         PASS           0.00           0.00          0.00  **
** example_tests.test_fifo_002         PASS           0.00           0.00          9.00  **
** example_tests.test_fifo_003         PASS           0.00           0.00         12.45  **
** example_tests.test_fifo_004         PASS           0.00           0.00         13.03  **
**************************************************************************************
** TESTS=4 PASS=4 FAIL=0 SKIP=0                       0.00           0.00          0.70  **
**************************************************************************************
```

TestFactory

cocotb.parametrize()

```
**************************************************************************************************************
** TEST                                                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
**************************************************************************************************************
** example_tests.test_fifo/valid_fraction=0.5/ready_fraction=0.5   PASS           0.00           0.00          0.00  **
** example_tests.test_fifo/valid_fraction=0.5/ready_fraction=1.0   PASS           0.00           0.00          8.47  **
** example_tests.test_fifo/valid_fraction=1.0/ready_fraction=0.5   PASS           0.00           0.00         16.98  **
** example_tests.test_fifo/valid_fraction=1.0/ready_fraction=1.0   PASS           0.00           0.00         19.42  **
**************************************************************************************************************
** TESTS=4 PASS=4 FAIL=0 SKIP=0                                                   0.00           0.02          0.13  **
**************************************************************************************************************
```

```python

@cocotb.test
@cocotb.parametrize(
    valid_fraction=[0.5, 1.0],
    ready_fraction=[0.5, 1.0],
)
async def test_fifo(
    dut,
    valid_fraction: float,
    ready_fraction: float,
) -> None:
    ...

```

COCOTB_TEST_FILTER

What?

- A regular expression filter for selecting tests
- Replacement for `TESTCASE` Makefile variable or `testcase` argument to `Runner.test()`

Why replace current methods?

- More expressive than current methods
- Works well with `cocotb.parametrize`
- Regular expressions are well understood

```python

@cocotb.test
@cocotb.parametrize(
    error_rate=[0.0, 0.01, 0.8],
    poll_rate=[50, 1000, 20000],
    resync_rate=[0, 100],
)
async def test_external_device(
    dut,
    error_rate: float,
    poll_rate: int,
    resync_rate: int
) -> None:
    ...

```

Run all parametrized tests

COCOTB_TEST_FILTER

```
(venv) ~/cocotb_examples
> make COCOTB_TEST_FILTER=test_external_device
```

TESTCASE

```
(venv) ~/cocotb_examples
> make TESTCASE=test_external_device/error_rate=0.0/poll_rate=50/resync_rate=0,example_tests.test_external_
e=0.0/poll_rate=20000/resync_rate=0,example_tests.test_external_device/error_rate=0.0/poll_rate=20000/resyn
e_tests.test_external_device/error_rate=0.01/poll_rate=1000/resync_rate=100,example_tests.test_external_dev
=0.8/poll_rate=50/resync_rate=100,example_tests.test_external_device/error_rate=0.8/poll_rate=1000/resync_r
```

cocotb

```python

@cocotb.test
@cocotb.parametrize(
    error_rate=[0.0, 0.01, 0.8],
    poll_rate=[50, 1000, 20000],
    resync_rate=[0, 100],
)
async def test_external_device(
    dut,
    error_rate: float,
    poll_rate: int,
    resync_rate: int
) -> None:
    ...

```

Run only tests where poll_rate == 50

```
(venv) ~/cocotb_examples
> make COCOTB_TEST_FILTER="test_external_device.*poll_rate=50"
```

```
**************************************************************************************************************************************
** TEST                                                                                STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
**************************************************************************************************************************************
** example_tests.test_external_device/error_rate=0.0/poll_rate=50/resync_rate=0      PASS           0.00           0.00          0.00  **
** example_tests.test_external_device/error_rate=0.0/poll_rate=50/resync_rate=100    PASS           0.00           0.00         10.16  **
** example_tests.test_external_device/error_rate=0.01/poll_rate=50/resync_rate=0     PASS           0.00           0.00         16.91  **
** example_tests.test_external_device/error_rate=0.01/poll_rate=50/resync_rate=100   PASS           0.00           0.00         19.24  **
** example_tests.test_external_device/error_rate=0.8/poll_rate=50/resync_rate=0      PASS           0.00           0.00         19.33  **
** example_tests.test_external_device/error_rate=0.8/poll_rate=50/resync_rate=100    PASS           0.00           0.00         19.24  **
**************************************************************************************************************************************
** TESTS=6 PASS=6 FAIL=0 SKIP=0                                                                    0.01           0.01          0.49  **
**************************************************************************************************************************************
```

Modeling Types

New Modeling Types

What are the new modeling types?

- Set of types to deal with common HDL values
  - `Logic`: 9-value logic scalar type
  - `Range`: right-bound-inclusive integer range
  - `Array`: `list`-like, arbitrarily-indexed, immutably-sized
  - `LogicArray`: Array of Logic + bitwise ops
- Replacement for `BinaryValue`

Why replace `BinaryValue`?

- Bugs
- Fundamentally assumes 0/1 values
- No indexing/slicing

Array

- Like Python `list`
- Immutable size
- Uses `Range` to describe arbitrary indexing

```python

# Deduced Range
>>> Array([1, "2", None, (1,)])
Array([1, "2", None, (1,)], Range(0, "to", 3))


# Explicit Range
>>> a = Array("1234", Range(56, "downto", 53))
>> a
Array (["1", "2", "3", "4"], Range(56, "downto", 53))


# Uses arbitrary indexing
>>> a[56]
"1"
>>> a[53]
"4"
>>> a[56:55]
Array(["1", "2"], Range(56, "downto", 55))
>>> a[53] = 170
>> a
Array(["1", "2", "3", 170], Range(56, "downto", 53))


# Works like list
>>> len(a)
4
>>> for elem in a:
...     print (elem)
"1"
"2"
"3"
170

```

LogicArray

- Similar to `Array`
- Constructs values into `Logic`
- Supports bitwise operations
- Convertible to/from `int`, `str`, `bytes`, sequences of `Logic`

```python

# Automatic conversion to Logic
>>> l = LogicArray([0, 1, 1, 0])
>>> l[3:1] = "100"
>>> l
LogicArray("1000", Range(3, "downto", 0))


# Bitwise operations
>>> l & LogicArray("01XZ")
LogicArray("00XX", Range(3, "downto", 0))
>>> l | LogicArray("0H0H")
LogicArray ("1101", Range(3, "downto", 0))
>>> ~l
LogicArray("0111", Range(3, "downto", 0))


# Conversion to/from integers
>>> LogicArray.from_signed (127, 8)
LogicArray("01111111", Range(7, "downto", 0))
>>> LogicArray("1000").to_unsigned()
8
>>> LogicArray("1000").to_signed()
-8


# Conversion to/from bytes
>>> LogicArray.from_bytes(b"\x0F\xF0", byteorder="big")
LogicArray("0000111111110000", Range(15, "downto", 0))
>>> LogicArray("01010101").to_bytes(byteorder="little")
b"U"

```

Incompatibilities

- `binaryRepresentation` removed
  - Use `from_signed`, `from_unsigned`, `to_signed`, `to_unsigned`
- `bigEndian` removed
  - Use `from_bytes`, `to_bytes`, takes byteorder
- Arithmetic operations removed
  - Convert to `int` first
  - Future work: `Signed` and `Unsigned` modeling types
- `COCOTB_RESOLVE_X` removed
  - Equality between ints and Xs no longer ValueError
  - Use `is_resolvable` to check for Xs before converting to int or bytes

Handles

Value Type Changes

What changed?

- Signal values are now LogicArray, not BinaryValue
- Array signal value are now Array, not list

Compatibilities

- Equality with existing types
- `.binstr`, `str()`
- `.signed_integer`, `.integer`, `int()`
- `.buff`
- `.is_resolvable`
- `bool()`, conditionals

```python

@cocotb.test
async def test_thing(dut) -> None:
    # .binstr, str()
    while dut.rst.value.binstr != "0":
        await Edge(dut.rst)

    # int(), .integer, .signed_integer
    PIPE_DEPTH = int(dut.PIPE_DEPTH.value)

    dut.input.value = 1
    for i in range(PIPE_DEPTH):
        # equality with int
        assert dut.output.value == 0
        await RisingEdge(dut.clk)

    # equality with str
    assert dut.output.value == "1"

```

Value Type Changes

Incompatibilities

- No more `BinaryValue`
- Indexing on arrayed signals uses HDL indexes

```python

@cocotb.test
async def test_thing(dut) -> None:
    #
    # logic [7:0] array_signal [3:0];
    #

    # Works
    assert dut.array_signal.value == [0, 1, 2, 3]

    # Breaks
    array_value = dut.array_signal.value
    assert array_value[0] == 0  # Fails, actual is 3

```

New [] Indexing Syntax

What?

- Get child objects in design hierarchy
- Alternative to dot syntax

Why?

- Dot syntax limited to Python identifiers
- May conflict with Python object attributes

When to use over dot syntax?

- Extended identifiers
- Name collisions

```python

@cocotb.test
async def test_thing(dut) -> None:

    # Works interchangeably with dot syntax
    dut["my_module"].my_signal

    # Python attribute '_log' shadows child signal
    dut.my_module["_log"]

    # Valid signal name is not valid Python
    dut my_module["my$signal"]
    dut.my_module["\\my extended identifier!\\"]

```

cocotb

Other Handle *Incompatibilities*

- Handle classes refactored and renamed
- Support for indexing into packed objects removed
  - Wasn’t consistently supported
  - Edge triggers weren’t consistent
  - Setting value wasn’t consistent
  - Index values instead
  - Or use unpacked objects

```python

@cocotb.test
async def test_thing(dut) -> None:

    async def drive_inputs() -> None:
        try:
            while True:
                await drive()
        finally:
            # Want this to run if the driver is cancelled
            drive_idle()

    # Drive inputs concurrently
    drive_task = cocotb.start_soon(drive_inputs())

    # Check 100 outputs
    for _ in range (100):
        await check()

    # Cancel task to stop driving input
    drive_task.cancel()

    ...  # test continues

```

Concurrency

cocotb.start_soon()

What is it?

- Way to run multiple independent threads of execution concurrently
- Replacement for `cocotb.fork()`

Why replace `cocotb.fork()`?

- Schedule re-entrancy causing bugs
- Inconsistent handling of Exceptions

```python

@cocotb.test
async def test_dff(dut) -> None:

    async def drive_inputs() -> None:
        dut.input.value = 0
        while True:
            await RisingEdge(dut.clk)
            dut.input.value = int(dut.input.value) + 1

    # Drive input concurrently to output checking
    cocotb start_soon(drive_inputs())

    # Check outputs
    for _ in range(100):
        prev_value = int(dut.output.value)
        await RisingEdge(dut.clk)
        assert dut.output.value = prev_value + 1

```

cocotb.start_soon()

Incompatibilities

- Runs *soon*, not immediately
- In those cases use `cocotb.start()`

```python

@cocotb.test
async def test_thing(dut) -> None:
    driver_running = False

    async def drive_inputs() -> None:
        nonlocal driver_running
        driver_running = True

    # Fails, 'drive_inputs' hasn't started.
    cocotb.start_soon(drive_inputs())
    assert driver_running

    # Passes, Forces 'drive_inputs' to start.
    await cocotb.start(drive_inputs())
    assert driver_running

```

Read more about the differences at https://fossi-foundation.org/blog/2021-10-20-cocotb-1-6-0.

`Task.cancel()`

What is it?

- Stops a `Task` from running
- `Task.cancel()` replaces `Task.kill()`
- Inspired by asyncio and trio

Why replace `Task.kill()`?

- Scheduler re-entrancy causes bugs
- Doesn't allow Tasks to "clean up"

How does it work?

- Schedules `CancelledError` to be thrown into cancelled Tasks
- Use context manager and try-finally blocks

```python

@cocotb.test
async def test_thing(dut) -> None:

    async def drive_inputs() -> None:
        try:
            while True:
                await drive()
        finally:
            # Want this to run if the driver is cancelled
            drive_idle()

    # Drive inputs concurrently
    drive_task = cocotb.start_soon(drive_inputs())

    # Check 100 outputs
    for _ in range (100):
        await check()

    # Cancel task to stop driving input
    drive_task.cancel()

    ...  # test continues

```

TaskManager

coming soon

What is it for?

- Structuring *asymmetric* concurrency
- Inspired by SV's fork/join, trio's nurseries, asyncio's TaskGroup
- Easy to use correctly
- Hard to use incorrectly

When to use it?

- When your test or components need to do multiple things at the same time.
- Use `cocotb.start_soon()` to run independent components (Drivers, Monitors, etc.)

```python

@cocotb.test
async def test_thing(dut) -> None:

    async with TaskManager() as tm:

        @tm. fork
        async def drive_input() -> None:
            for _ in range (100):
                await drive()

        @tm. fork
        async def check_output() -> None:
            for _ in range (100):
                await check()

    ...  # joins all when context ends

```

TaskManager

- Create child Tasks
  - `@fork` decorator
  - `start_soon`, `start`
- Implicit “Join all” if context ends
- `await` Tasks
- Cancel Tasks
- Catch Exceptions in Tasks

coming soon

```python

@cocotb.test
async def test_thing(dut) -> None:
    async with TaskManager() as tm:

        @tm.fork
        async def drive_input() -> None:
            while True:
                await drive()

        # start with start_soon
        drive_task = tm. start_soon(drive_input())

        # start with fork decorator
        @tm.fork
        async def check_output() -> None:
            for _ in range (100):
                await check()

        try:
            # wait for child Task to finish
            await check_output
        except Exception:
            # handle Exception from child Task
            ...

        # cancel child Task
        drive_task.cancel()

    # all tasks are done by now

```

TaskManager

- Use ExceptionGroups
- Use `except*` syntax from Python 3.11

```python

@cocotb.test
async def test_thing(dut) -> None:
    try:
        async with TaskManager() as tm:

            @tm.fork
            async def drive_input() -> None:
                # stuff
                raise ValueError  # oops

            @tm.fork
            async def check_output() -> None:
                # various things are occuring
                raise IndexError  # oops

    except* ValueError:
        # handle ValueError from 'drive_input'
        # or any other child Task
        ...

    except* IndexError:
        # handle IndexError from 'check_output'
        # or any other child Task
        ...

```

coming soon

TaskManager

- Canceling recursively cancels children Tasks
- Prevents leaking Tasks

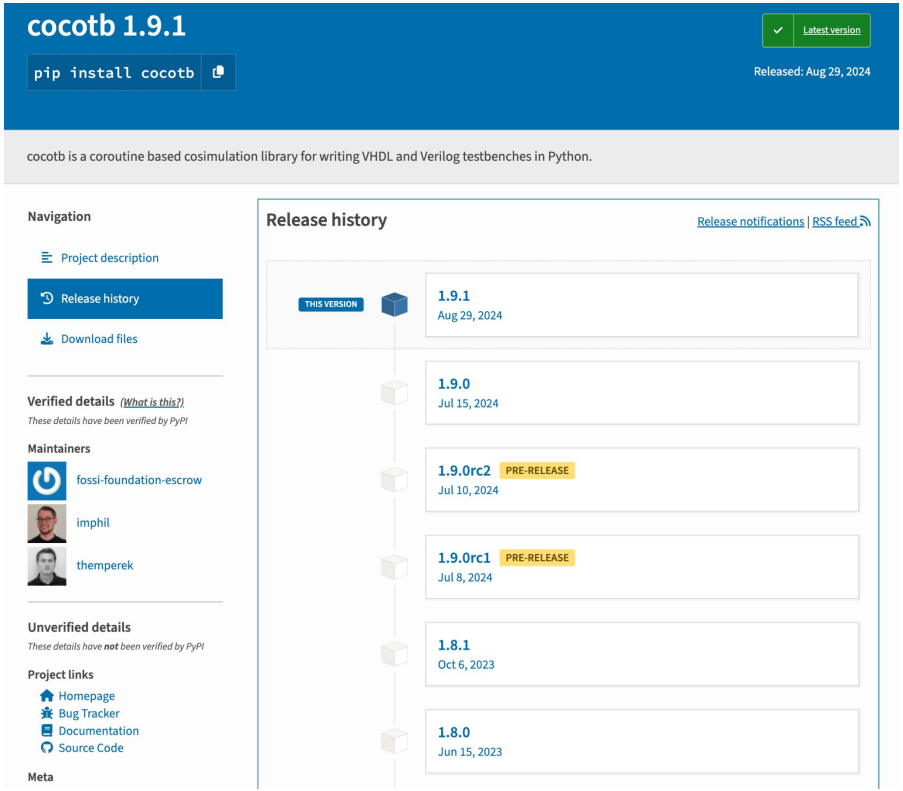
coming soon

```python

@cocotb.test
async def test_thing(dut) -> None:

    async def do_stuff() -> None:

        async with TaskManager() as tm:

            @tm.fork
            async def drive_input() -> None:
                try:
                    ...
                finally:
                    # TaskManager cancels child Tasks
                    # when it gets CancelledError
                    print("Cleaned up!")

            @tm.fork
            async def check_output() -> None:
                ...

    # do_stuff in Task so we can cancel it
    do_stuff_task = cocotb.start_soon(do_stuff())

    # Cancel Task with running TaskManager
    do_stuff_task. cancel()

```

Additional Changes

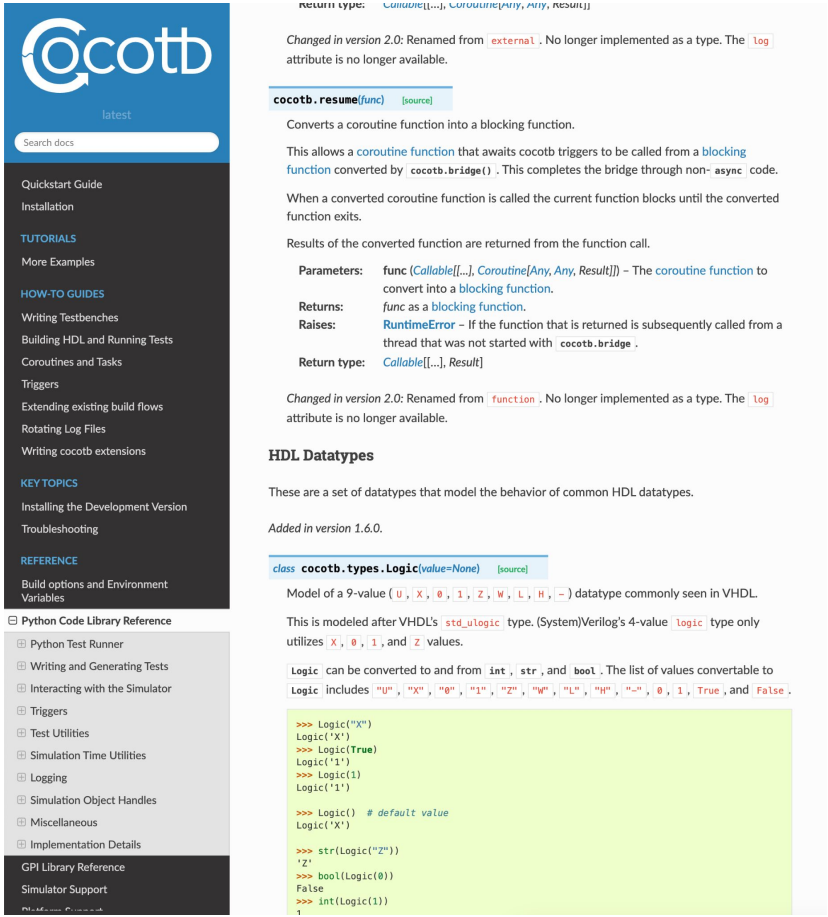
Public API Guarantees

- Properly define Python public API using Python idioms
- Can make guarantees about stability of API
- Semantic Versioning



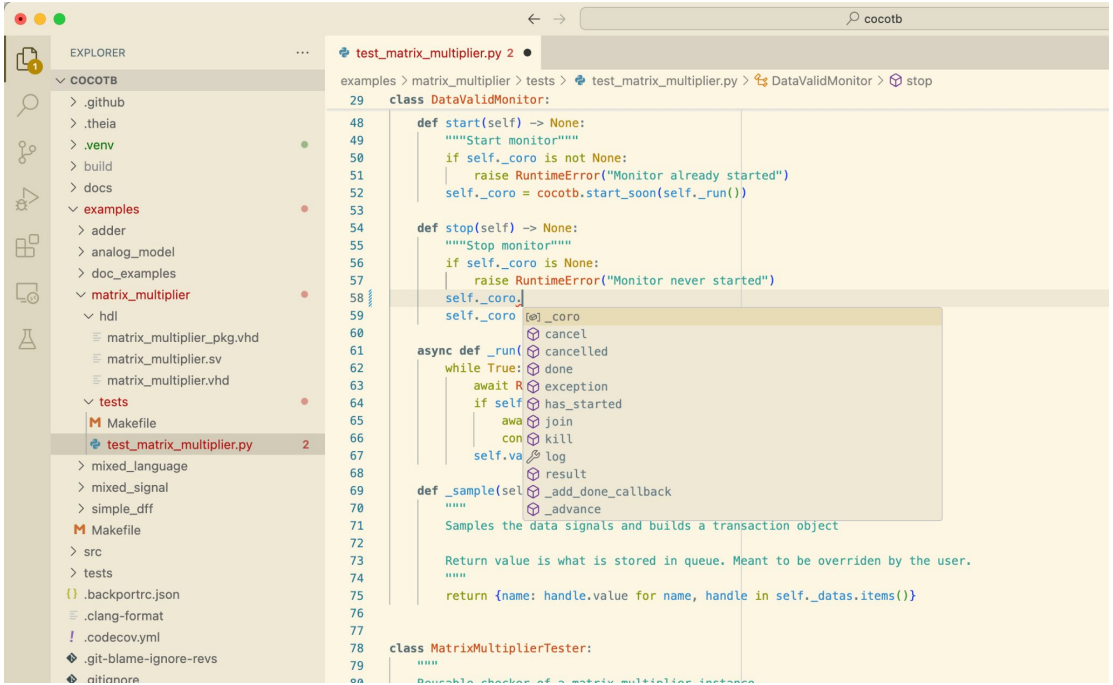
Documentation

- API reference documentation complete
- If it’s documented, it’s part of the public API
- Guaranteed by running pydocstyle in CI



Typing

- Check your cocotb tests with type checkers
- Better suggestions in editors
- Guaranteed by running mypy in CI



Migrating to cocotb 2.0

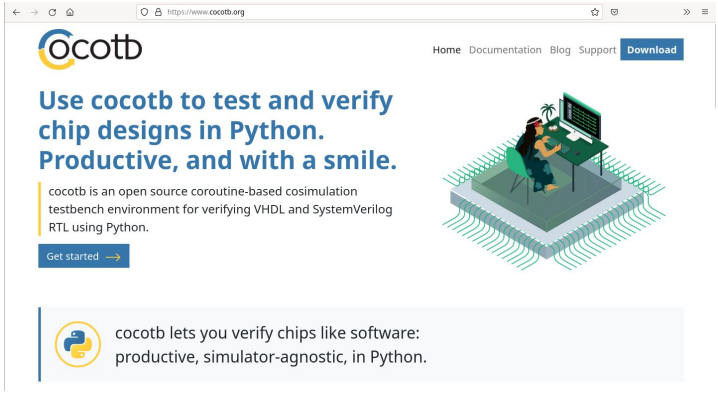
1. Upgrade to the latest cocotb 1.x.
2. Resolve all deprecation warnings.
3. Upgrade to cocotb 2.0 (or a current master version)
4. Fix remaining issues.

> A migration guide will be released together with cocotb 2.0. Tell us your migration experience by opening a GitHub Discussion item at https://github.com/cocotb/cocotb/discussions.

```
/path/to/02.fifo/test_fifo.py:111: DeprecationWarning:
Using `task` directly is prefered to `task.join()` in all
situations where the latter could be used.`
 await read_thread.join()
 2025.00ns DEBUG    cocotb.fifo
WRITE: Wait for 1 clock cycles
 2025.00ns DEBUG    cocotb.fifo
WRITE: wait
 2025.00ns DEBUG    cocotb.fifo
READ: Wait for 0 clock cycles
/path/to/site-packages/cocotb/types/logic_array.py:807:
FutureWarning: The behavior of bool casts and using
LogicArray in conditionals may change in the future. Use
explicit comparisons (e.g. `LogicArray() == "11"`) instead
 warnings.warn(
 2025.00ns DEBUG    cocotb.fifo
READ: FIFO empty, not reading
 2025.00ns DEBUG    cocotb.fifo
READ: Wait for 3 clock cycles
 3025.00ns DEBUG    cocotb.fifo
WRITE: wait for falling clock edge
 4025.00ns DEBUG    cocotb.fifo
WRITE: Wrote word 1 to FIFO, value 0x0
/path/to/02.fifo/fifo_test_common.py:134:
DeprecationWarning: `.integer` property is deprecated. Use
`value.to_unsigned()` instead.
```

Remember

- Verification is software.
- cocotb 2.0 is around the corner. Start testing it now!
- cocotb is not just verification in Python – it can turbo-charge your chip development.



www.cocotb.org
GitHub: cocotb
LinkedIn: cocotb
Twitter/X: @cocotbnews