cocotb 2.x and older tutorials

This page answers: *I copied a cocotb example off the internet and it does not work, or it works but prints warnings. Which of us is wrong?*

Neither of you. The example is right for cocotb 1.x, but this course runs cocotb 2.0.1, and the 2.0 release renamed and removed a lot of the 1.x interface.

Almost every cocotb tutorial, blog post, Stack Overflow answer, chatbot completion and university lab handout was written against 1.x. Some of those examples still run on 2.0.1, and they are the dangerous ones, because they give you no reason to suspect them. That's why this page starts with them.

We ran everything below in the workbench and pasted the output unedited. The four demonstration files ship in `labs/cocotb-intro/oldapi/`, and you'll run them in cocotb on a counter, Part 3.

Forms that still work

`units=` instead of `unit=`

`units=` is the 1.x form you'll come across most often, and cocotb 2.0.1 still accepts it:

```
cocotb.start_soon(Clock(dut.clk, 10, units="ns").start())
await Timer(20, units="ns")
```

```
        0.00ns  WARNING   py.warnings     /work/labs/cocotb-intro/oldapi/test_1x_units.py:24:
DeprecationWarning: The 'units' argument has been renamed to 'unit'.
                                  cocotb.start_soon(Clock(dut.clk, 10, units="ns").start())
...
    41.00ns INFO     cocotb.regression   test_1x_units.the_old_spelling_still_runs passed
** TESTS=1 PASS=1 FAIL=0 SKIP=0                          41.00          0.00     19633.12  **
```

The test passes. The warning is a single `WARNING` line among the INFO lines, so it scrolls past, and nothing else in the run tells you that this code will stop working. A deprecation means the old spelling is going to be removed in a later release, so write `unit=`.

`.integer`, `.binstr`, `.signed_integer`, `.buff`

These value getters behave the same way: they still work and they warn, and each warning names the replacement to use.

```
      41.00ns  WARNING   py.warnings     DeprecationWarning: `logic_array.integer` getter is
deprecated. Use `logic_array.to_unsigned()` instead.
    41.00ns INFO     cocotb.counter   v.integer = 42
      41.00ns  WARNING   py.warnings     DeprecationWarning: `logic_array.binstr` getter is
deprecated. Use `str(logic_array)` instead.
    41.00ns INFO     cocotb.counter   v.binstr  = 00101010
```

1.x	2.x
`sig.value.integer`	`int(sig.value)` or `sig.value.to_unsigned()`
`sig.value.binstr`	`str(sig.value)`
`sig.value.signed_integer`	`sig.value.to_signed()`

1.x	2.x
`sig.value.buff`	`sig.value.to_bytes(byteorder="big")`

`sig.value.is_resolvable` is not deprecated, and it's the right way to ask whether a value contains any `X` before you convert it.

`MODULE` and `TOPLEVEL` in the Makefile

1.x makefiles say:

```
TOPLEVEL = counter
MODULE   = test_tiny
```

Under 2.0.1's `Makefile.sim` these still work, with one warning:

```
/usr/local/lib/python3.12/dist-packages/cocotb_tools/makefiles/simulators/
Makefile.icarus:66: Using MODULE is deprecated, please use COCOTB_TEST_MODULES instead.
...
** TESTS=1 PASS=1 FAIL=0 SKIP=0                  1.00          0.00       1178.18  **
```

The 2.x names are `COCOTB_TOPLEVEL` and `COCOTB_TEST_MODULES`. Use those, as DD104's `sim.mk` does.

> The translation from the old names to the new ones is done by cocotb's makefiles, not by cocotb itself. If you drive a simulation without the makefiles (with the Python runner, `pytest` or a hand-written `vvp` command line), only `COCOTB_TEST_MODULES` is read, and `MODULE` is ignored without any warning.

`COCOTB_TESTCASE`

`COCOTB_TESTCASE` is deprecated in favour of `COCOTB_TEST_FILTER`, which takes a regular expression:

```
     0.00ns WARNING  py.warnings   DeprecationWarning: COCOTB_TESTCASE is deprecated in favor
of COCOTB_TEST_FILTER
```

Read A filter that runs nothing before you use either.

Forms that fail loudly

These produce a traceback with a line number, so once you have seen each one you'll be able to fix it in seconds.

`cocotb.fork` → `cocotb.start_soon`

```
cocotb.fork(Clock(dut.clk, 10, unit="ns").start())
```

```
    0.00ns WARNING  ..b.Test fork_is_gone.fork_is_gone  module 'cocotb' has no attribute 'fork'
                                                         Traceback (most recent call last):
                                                        File "/work/labs/cocotb-intro/oldapi/
test_1x_fork.py", line 22, in fork_is_gone
                                                               cocotb.fork(Clock(dut.clk, 10,
unit="ns").start())   # AttributeError
                                                            ^^^^^^^^^^^
```

```
                                                    AttributeError: module 'cocotb' has
no attribute 'fork'
```

The replacement is `cocotb.start_soon(...)`, with the same argument. (There is also `cocotb.start(...)`, which is the version you `await`; it lets the new task run before your code continues.)

`dut.sig <= value` → `dut.sig.value = value`

1.x let you assign to a signal with `<=`, the way you would in Verilog:

```
dut.rst_n <= 1
```

```
     0.00ns WARNING  ..re.le_is_not_assignment_any_more   '<=' not supported between instances
of 'LogicObject' and 'int'
                                                     Traceback (most recent call last):
                                                    File "/work/labs/cocotb-intro/oldapi/
test_1x_assign.py", line 25, in le_is_not_assignment_any_more
                                                      dut.rst_n <= 1         # TypeError
                                                        ^^^^^^^^^^^^^^
                                                    TypeError: '<=' not supported between
instances of 'LogicObject' and 'int'
```

A `TypeError` about `<=` reads like a bug in a comparison, because to Python the line is a comparison. It means that the author used the old assignment form, which 2.0 removed.

`cocotb.binary` and `cocotb.result`

```
from cocotb.binary import BinaryValue
from cocotb.result import TestFailure
```

These fail while cocotb is importing your file, before any test runs, so there is no test name in the output and no results table:

```
  (this one never reached 'Running tests' -- it failed while cocotb was
   still importing the file, so no test ran at all)
Traceback (most recent call last):
  ...
  File "/work/labs/cocotb-intro/oldapi/test_1x_imports.py", line 20, in <module>
    from cocotb.binary import BinaryValue     # ModuleNotFoundError, at IMPORT time
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'cocotb.binary'
```

`cocotb.result` is harder to spot. The module still exists but is now empty, so the error is about a missing name rather than a missing module, and it reads like a typo:

```
AttributeError: module 'cocotb.result' has no attribute 'TestFailure'
```

1.x	2.x
`from cocotb.binary import BinaryValue`	`from cocotb.types import LogicArray`

1.x	2.x
`raise TestFailure("...")`	`assert cond, "..."`
`raise TestSuccess()`	`cocotb.pass_test()`, or just `return`
`@cocotb.coroutine` + `yield`	`async def` + `await`
`cocotb.regression.TestFactory`	`@cocotb.parametrize(...)` (`TestFactory` still exists)

The second traceback

When cocotb fails during discovery (a missing module, an import error, no tests found), it prints a second traceback after the real one:

```
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/dist-packages/cocotb/_init.py", line 117, in _sim_event
    cocotb._regression_manager._fail_simulation(msg)
     File   "/usr/local/lib/python3.12/dist-packages/cocotb/regression.py",   line   899,   in
_fail_simulation
    self._running_test.abort(self._sim_failure)
    ^^^^^^^^^^^^^^^^^^
AttributeError: 'RegressionManager' object has no attribute '_running_test'
     0.00ns ERROR    gpi   Passing event to upper layer failed
```

You can ignore this one. It's cocotb's own error handler failing while it reports your error, because there was no running test for it to abort. The real problem is in the traceback above it. You can tell the two apart because `RegressionManager`, `_running_test`, `gpi` and `_sim_event` are not names from your code.

A filter that runs nothing

None of the forms above will cost you a day, but this one can.

`make nothing` in `labs/cocotb-intro` runs the real six-test suite with two letters swapped in the test filter:

```
COCOTB_TESTCASE = chekcing
```

```
     0.00ns INFO     cocotb   Initialized cocotb v2.0.1 from /usr/local/lib/python3.12/dist-
packages/cocotb
     0.00ns WARNING  py.warnings   DeprecationWarning: COCOTB_TESTCASE is deprecated in favor
of COCOTB_TEST_FILTER
     0.00ns INFO     cocotb   Running tests
     0.00ns WARNING  cocotb.regression   No tests left after filtering with: chekcing$

   exit status 0, no FAIL anywhere, and zero tests ran.
```

There is no results table, so there's no `TESTS=`, no `PASS=` and no `FAIL=`. The whole run, including the lines not shown above, is six INFO lines, two WARNINGs and a clean exit, which in CI shows up as a green tick.

`results.xml` does record what happened, if anyone reads it:

```
<testcase name="load_then_count" classname="test_03_checking" ... >
  <skipped />
</testcase>
```

All six tests are marked `<skipped/>`, and none of them ran.

If you take one habit from this page, make it this one: after every cocotb run, read the `TESTS=` line and check that the count matches the number of tests you wrote. The colour of the output and the exit code won't tell you; only the count will. A cocotb run that tested nothing looks just like one that tested everything, apart from one missing line.

This is DD104's argument on a small scale. SimProc shipped with one test that passes on thirteen broken processors, and a filter typo passes on all thirteen and on the good one too. That's why "My tests pass" doesn't count as a result here, while "Six tests ran, they passed on the good core, and this one fails on the broken core" does.

Conversion table

You found (1.x)	Write this (2.0.1)	What happens if you don't
`Timer(1, units="ns")`	`Timer(1, unit="ns")`	Works, `DeprecationWarning`
`Clock(clk, 10, units="ns")`	`Clock(clk, 10, unit="ns")`	Works, `DeprecationWarning`
`Clock(clk, 10)`	`Clock(clk, 10, unit="ns")`	Works, as a 10 ps clock
`cocotb.fork(coro)`	`cocotb.start_soon(coro)`	`AttributeError: module 'cocotb' has no attribute 'fork'`
`dut.sig <= 1`	`dut.sig.value = 1`	`TypeError: '<=' not supported between instances of 'LogicObject' and 'int'`
`dut.sig = 1`	`dut.sig.value = 1`	`AttributeError: ... Did you forget to add `.value`?` (silent in 1.x)
`sig.value.integer`	`int(sig.value)`	Works, `DeprecationWarning`
`sig.value.binstr`	`str(sig.value)`	Works, `DeprecationWarning`
`sig.setimmediatevalue(v)`	`sig.value = Immediate(v)`, from `cocotb.handle`	Works, `DeprecationWarning`
`from cocotb.binary import BinaryValue`	`from cocotb.types import LogicArray`	`ModuleNotFoundError` at import
`raise TestFailure("x")`	`assert cond, "x"`	`AttributeError` on `cocotb.result`
`@cocotb.coroutine` / `yield`	`async def` / `await`	`AttributeError: module 'cocotb' has no attribute 'coroutine'`
`from cocotb.triggers import Join`	`from cocotb.task import Join`	Works, `DeprecationWarning`
`MODULE=`	`COCOTB_TEST_MODULES=`	Works in makefiles and warns; ignored elsewhere

You found (1.x)	Write this (2.0.1)	What happens if you don't
`TOPLEVEL=`	`COCOTB_TOPLEVEL=`	Works, warns
`TESTCASE=` / `COCOTB_TESTCASE=`	`COCOTB_TEST_FILTER=` (a regex)	Works, warns

Ask the installed cocotb

The installed version's own function signatures are the only reference that is guaranteed to match the cocotb you are running. `inspect.signature(cocotb.test)`, for example, prints the keywords that the `@cocotb.test` decorator accepts:

```
(obj: Union[Callable[..., Coroutine[cocotb._base_triggers.Trigger, NoneType, NoneType]],
 cocotb._decorators.Parameterized, NoneType] = None, *, timeout_time: Optional[float] = None,
 timeout_unit: Literal['step', 'fs', 'ps', 'ns', 'us', 'ms', 'sec'] = 'step',
     expect_fail:     bool     =     False,     expect_error:     Union[Type[BaseException],
Tuple[Type[BaseException],
 ...]] = (), skip: bool = False, stage: int = 0, name: Optional[str] = None) -> ...
```

(We wrapped it here to fit the page; it comes out as one long line.) If a keyword you read about somewhere isn't in that list, it's from 1.x. You'll run this check yourself in the lab.

Check yourself

1. A tutorial's test runs, passes and prints `DeprecationWarning: The 'units' argument has been renamed to 'unit'.` Is anything wrong with your result?
2. A run prints no results table and exits 0. What happened?
3. `TypeError: '<=' not supported between instances of 'LogicObject' and 'int'`. What did the author of that line mean to write?

Next: Driving and checking a design.