DOULOS KnowHow WEBINARS

Delivering KnowHow www.doulos.com

Class Based Design Verification with Python+coctob

Verification Futures 2025

Presenter: Matt Bridle
Certified Training Instructor

Class Based Design Verification with Python+cocotb

Agenda

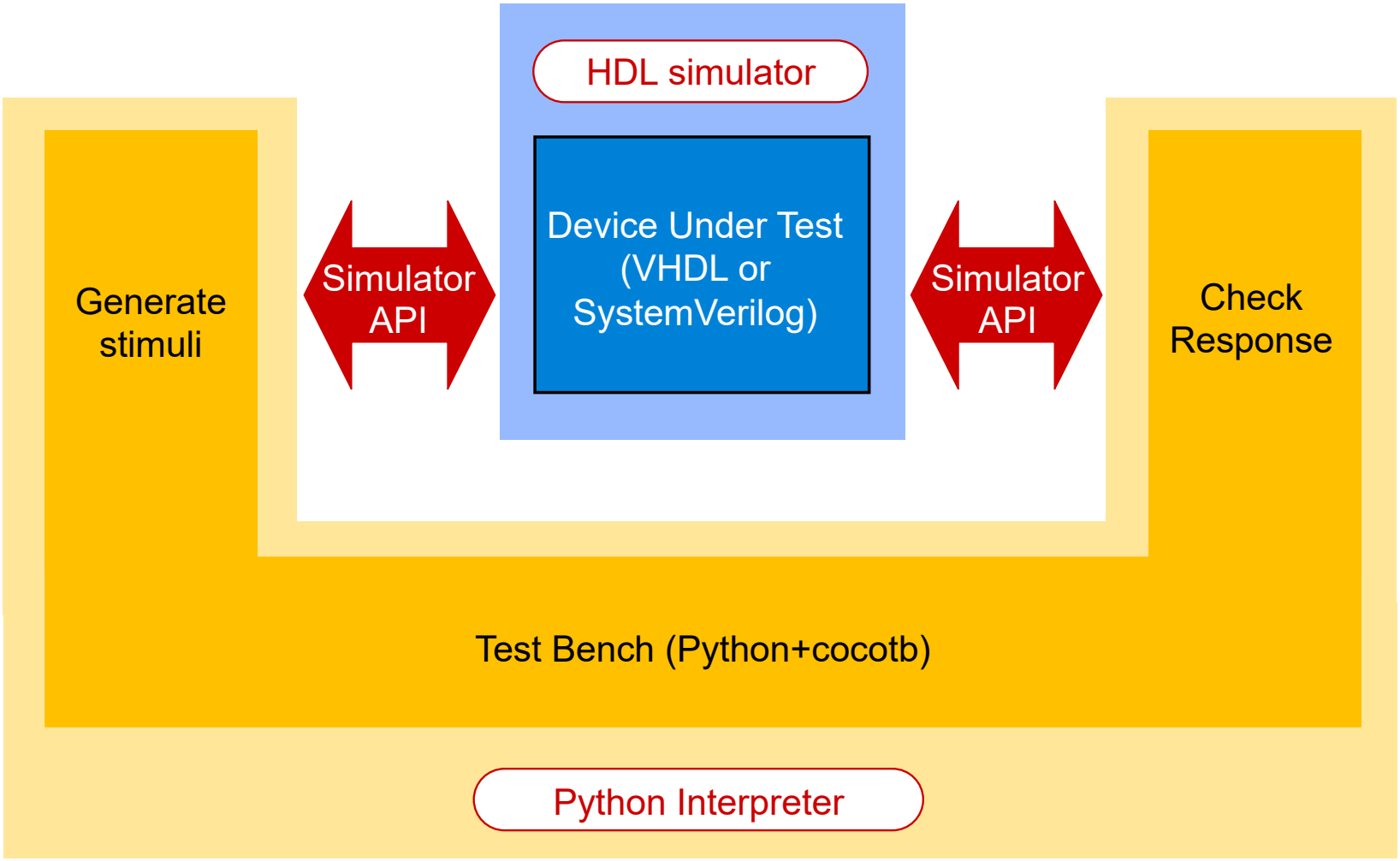
- Introducing cocotb
- Structuring a verification framework
- A reusable cocotb verification solution

DOULOS

Introducing cocotb

- *Co*-routine-based *Co*-simulation *T*est *B*ench environment
- Created by Philipp Wagner and others
- Extends Python to support hardware verification
  - Enables reading and writing of HDL design elements
  - Handles concurrency and synchronization elegantly
  - Makes it easier to create verification components
- Open-source (BSD licensed)
- Works with a growing number of commercial and free simulators
- www.cocotb.org

Co-simulation with cocotb



Advantages of cocotb

- Why would we choose cocotb for verification?
  - Many engineers are already familiar with Python
  - No need to learn a heavyweight verification methodology
  - Support for many industry-standard simulators
  - Python offers a huge number of packages for many applications
    - Can be used to generate stimulus
    - … and reference models

A Simple cocotb Test

```
import cocotb
from cocotb.triggers import Timer


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

    # Set input to zero
    dut.IN1.value = 0

    await Timer(10, units="ns")

    assert dut.OUT1.value == 1
```

Mark as a test

Assign 0 to port IN1

Wait for 10 ns

Check that port OUT1 has the value 1

Running a cocotb test

- Configure a test using a Makefile
  - … and run **`make`**
- Or using a *test runner*
  - … and run **`pytest`**
- Using standard Python logging module for formatted output

```
0.00ns INFO         cocotb.my_top_level                Starting test
```

- Reports a test summary at the end

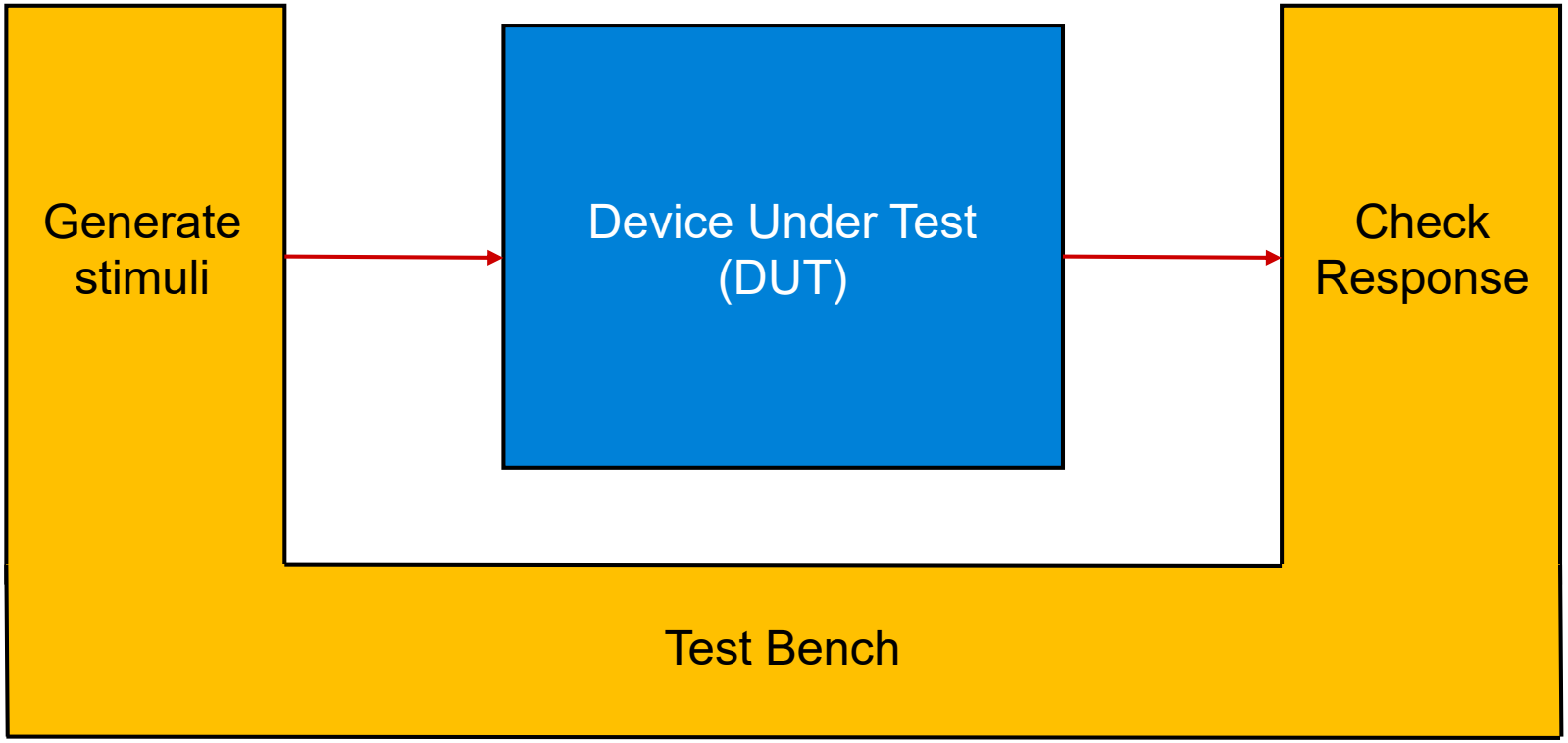
```
**************************************************************************************
** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s)   **
**************************************************************************************
** my_tester.my_tester           PASS         170.00           0.00      57843.42   **
**************************************************************************************
** TESTS=1 PASS=1 FAIL=0 SKIP=0                170.00           0.03       6504.67   **
**************************************************************************************
```

Class Based Design Verification with Python+cocotb

Agenda

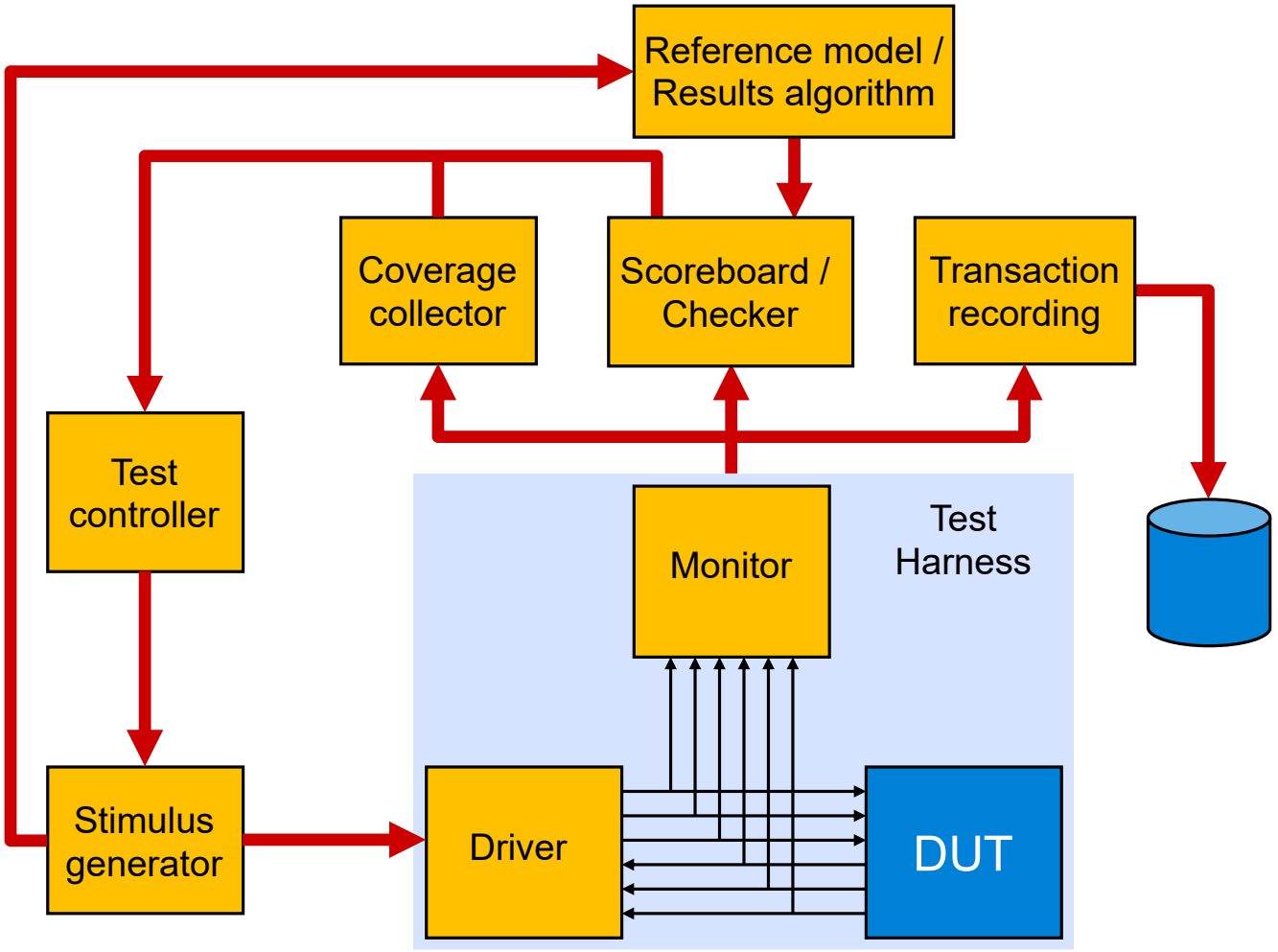
- Introducing cocotb
- ➡ Structuring a verification framework
- A reusable cocotb verification solution

A Basic Test Bench



- The classic all-in-one test bench does not scale well
  - Okay for block-level testing
  - No good for complex verification tasks
    - Difficult to configure in order to test different features

Reusable Verification Framework



Much better – but can we do it in cocotb?

Requirements on cocotb

- We need a tidy way to implement…
  - Verification components
  - Transactions
    - With randomization for stimulus generation
  - Communication channels
- We need concurrency and synchronization
  - Hardware is naturally parallel
  - Each verification component needs its own thread
- Verification IP for well-used bus protocols would be nice

Class Based Design Verification with Python+cocotb

Agenda

- Introducing cocotb
- Structuring a verification framework
- ➡ A reusable cocotb verification solution

DOULOS

Concurrency in cocotb

- Simulation involves concurrency
  - cocotb uses Python coroutines (asynchronous functions) to achieve this
  - cocotb Triggers and Events enable synchronization
  - cocotb Queue enables message passing

```
async def set_inputs_and_wait(dut,val):
    dut.A.value = val
    dut.B.value = val

    await Timer(5, units="ns")


@cocotb.test()
async def my_test(dut):
    await set_inputs_and_wait(dut, 1)
    check(dut.F.value, 0)

    await set_inputs_and_wait(dut, 0)
    check(dut.F.value, 1)
```

Coroutine

Set device inputs

Wait for timer

Coroutine must be **awaited** (not called)

Example - APB Transaction Class

```
class APB_trans:
    next_id = 0
    APB_ADDR_WIDTH = 8
    APB_DATA_WIDTH = 32

    def __init__(self, addr = 0, data = 0, is_write = False):
        self.id = APB_trans.next_id
        APB_trans.next_id += 1
        self.addr = addr
        self.data = data
        self.is_write = is_write

    def rand(self):
        self.addr  = random.randrange(0, 2**APB_trans.APB_ADDR_WIDTH)
        self.data  = random.randrange(0, 2**APB_trans.APB_DATA_WIDTH)
        self.is_write = random.choice([True, False])
        return self

    def get_is_write(self): return self.is_write
    def get_data(self): return self.data
    def get_addr(self): return self.addr
    def psprint(self):
        return f'({self.id}, {self.addr}, {self.data}, {self.is_write})'
```

Random data generation using standard Python functions

Example – APB Monitor Class

```
class APB_monitor:
    def __init__(self,dut,chan):
        self.dut = dut
        self.chan = chan

    async def body(self):
        self.dut._log.debug(f'Starting monitor body')
        while True:
            await RisingEdge(self.dut.PCLK)

            if self.dut.PSEL.value == 1 and self.dut.PENABLE.value == 1:

                addr = self.dut.PADDR.value
                if self.dut.PWRITE.value == 1:
                    write = True
                    data = self.dut.PWDATA.value
                else:
                    write = False
                    data = self.dut.PRDATA.value

                t = APB_trans(addr, data, write)

                self.dut._log.debug(f'Queueing monitored tx: {t.psprint()}')
                self.chan.put_nowait(t)
```

Wait for rising clock edge

Create a transaction object

Put transaction into channel (a cocotb Queue)

IP from cocotb-bus

- Bus protocol support package for cocotb
- Contains base classes for many verification components
  - Driver
  - Monitor
  - Scoreboard
  - Queue
  - …etc
- cocotb-bus also contains prebuilt components for some buses
  - AMBA, Avalon, XGMII and others
  - Support varies between components

Example – APB Driver Class (1)

```
class APB_driver(Driver):

    def __init__(self,dut,chan):
        self.dut  = dut
        self.chan = chan

    async def _driver_send(self, t, sync=True, **kwargs):
        self.dut._log.debug(f'Driving {t.psprint()}')

        self.dut.PSEL.value    = 1
        self.dut.PENABLE.value = 0
        self.dut.PADDR.value   = t.get_addr()
        self.dut.PWRITE.value  = t.get_is_write()

        if t.get_is_write():
            self.dut.PWDATA.value = t.get_data()

        await RisingEdge(self.dut.PCLK)

        self.dut.PENABLE.value = 1

        await RisingEdge(self.dut.PCLK)

        self.dut.PSEL.value    = 0
        self.dut.PENABLE.value = 0
        self.dut.PWRITE.value  = 0
```

Inherit from cocotb-bus Driver class

Override base class _driver_send method

Example – APB Driver Class (2)

```
class APB_driver(Driver):
    …
    …

    async def body(self):
        self.dut._log.debug("Starting driver body")
        next_trans = await self.chan.get()
        try:
            while True:
                self.dut._log.debug(f'About to drive {next_trans.psprint()}')
                await self.send(next_trans)
                next_trans = self.chan.get_nowait()

        except asyncio.QueueEmpty:
            self.dut._log.debug("Stimulus exhausted")
```

Blocking get

Non-blocking get

Exception indicating empty queue

Example – Top-level APB Class

```
class APB_env:

    def __init__(self,dut,num_trans):
        self.stim_chan = Queue(2)
        self.mon_chan  = Queue()
        self.finish    = Event()
        self.gen = APB_stream(dut,self.stim_chan,num_trans,APB_trans())
        self.bfm = APB_driver(dut,self.stim_chan)
        self.mon = APB_monitor(dut,self.mon_chan)
        self.chk = APB_checker(dut,self.mon_chan,num_trans,self.finish)
        dut._log.debug('Completed APB_env constructor')

    async def run(self):
        await cocotb.start(self.gen.body())
        await cocotb.start(self.mon.body())
        await cocotb.start(self.chk.body())
        await cocotb.start(self.bfm.body())
        await self.finish.wait()
```

Channels for driver and monitor

Start each verification component in its own thread

cocotb Event object used to synchronize threads

Random Stimulus and Coverage

- Python rand package can generate random data
  - Many distributions available (uniform, normal, gaussian etc.)
  - Can weight random choices

- See also
  - https://constrainedrandom.readthedocs.io/en/latest/intro.html
  - https://py-vsc.readthedocs.io/en/latest/introduction.html

Conclusion

- To implement…
  - Verification components **– use Python classes**
  - Transactions **– use Python classes**
    - With randomization for stimulus generation **– use Python random**
  - Communication channels **– use cocotb Queues**
- For concurrency and synchronization
  - Hardware is naturally parallel **– use coroutines**
  - For verification component threads **– use coroutines and cocotb Events**
- Some Verification IP is available **– in cocotb-bus**

- **Conclusion**
  - We can have the advantages of Python and cocotb
  - … and we don't need to compromise the robustness of the methodology

Thank you for attending

- Any Questions…?

DOULOS

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » AMD » Intel FPGA
Embedded Software & Arm	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Android » Arm » Rust » Zephyr
Python, AI & Machine Learning	» Python » Edge AI » Deep Learning

SystemVerilog OUT OF THE BOX · UVM · SYSTEMC™ · yocto PROJECT PARTICIPANT · Zephyr · SECURE EMBEDDED · APPROVED TRAINING PARTNER arm · AI & MACHINE LEARNING