CPE 470 - CocoTB

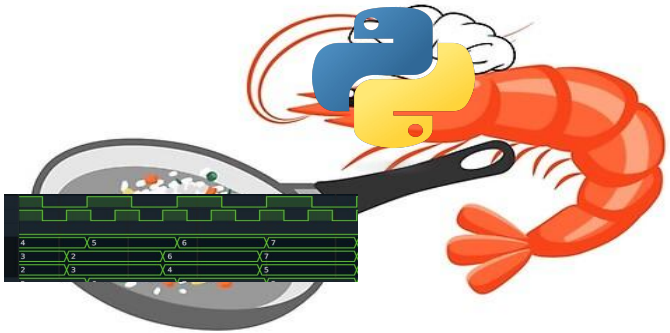
Verification Challenges

- Verification is inherently software
  - Runs only in simulator
- Verilog / System Verilog originally designed as HDL
  - Specialized for describing hardware, not software
- System Verilog does have OOP features
  - Steep learning curve
  - Less modernized

System Verilog Class

```
class myPacket;
    bit [2:0]  header;
    bit        encode;
    bit [2:0]  mode;
    bit [7:0]  data;
    bit        stop;

    function new (bit [2:0] header = 3'h1, bit [2:0] mode = 5);
        this.header = header;
        this.encode = 0;
        this.mode   = mode;
        this.stop   = 1;
    endfunction

    function display ();
        $display ("Header = 0x%0h, Encode = %0b, Mode = 0x%0h, Stop = %0b",
                  this.header, this.encode, this.mode, this.stop);
    endfunction
endclass
```

YOU'RE TELLING ME



A PYTHON TESTED THIS BENCH

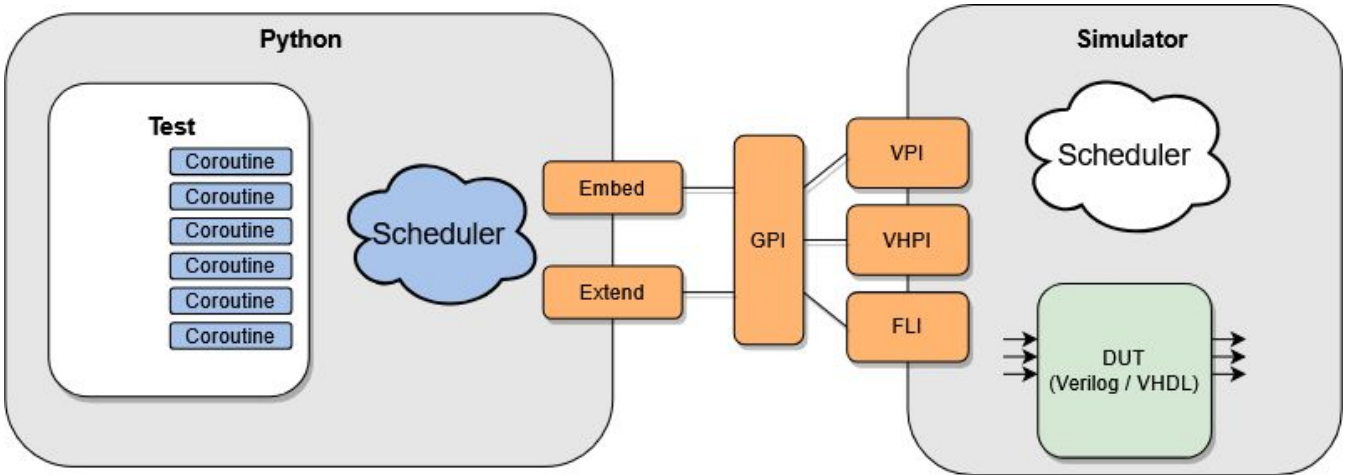
Coco TB

Glossary

CocoTB: Coroutine Co-Simulation Test Bench

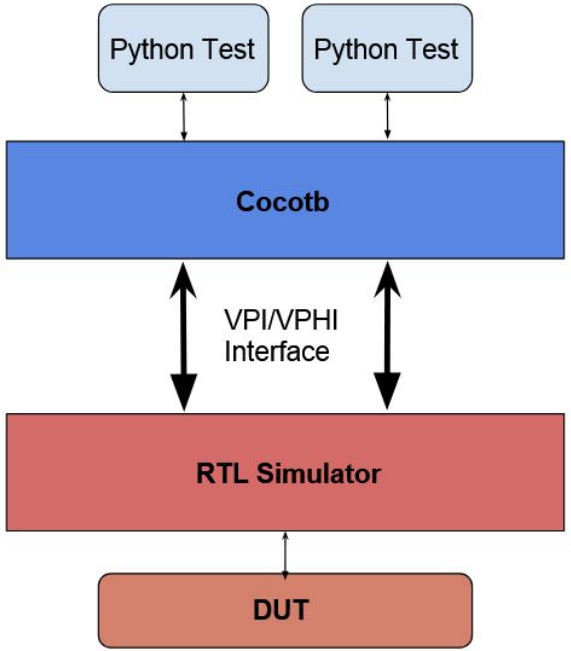
DUT: device under test, the top-level thing being tested

- **CocoTB** introduces Python-based testbench development
  - It is NOT a simulator! It interacts with a simulator
  - Uses verilator, icarus, or industry-grade simulator under the hood
- Takes turns between python and simulator
  - Python Code Introduces New Data to **DUT**
  - Waits while simulator executes a time step
  - Back and forth between python and simulator of choice



CocoTB Structure

- Use CocoTB in addition to (or instead of) verilog test bench
  - Often useful to have both kinds
- Write multiple python tests
  - Every time tests hit a delay or clock event, will defer to simulator
  - After event, simulator sends results back to python
- Abstract common operations into python classes
  - Create more easily reusable blocks to drive inputs and monitor outputs



Setting Up a Test

- Each test configured with a Makefile
  - Abstracts away the build process for different simulators
  - Easy to switch between them
- Somewhat Language Agnostic
  - Could bring same testbench to VDHL
  - Only switch lang and simulator
- Can turn waves on and off
  - Often don't want waves for sufficiently large tests

```
TOPLEVEL_LANG ?= verilog
SIM ?= icarus

VERILOG_SOURCES = top.sv  # List of Source Files
# Folders to look for source files
VERILOG_INCLUDE_DIRS = rtl

COCOTB_TOPLEVEL     := top       # CocoTB Version 2
TOPLEVEL := $(COCOTB_TOPLEVEL)      # CocoTB Version 1
COCOTB_TEST_MODULES := top_test # CocoTB Version 2
MODULE := $(COCOTB_TEST_MODULES)    # CocoTB Version 1

WAVES := 1  # Turn on waveforms

# Call CocoTB's makefile
include $(shell cocotb-config --makefiles)/Makefile.sim
```

Interacting With DUT

- Writing a Signal:

```
dut.signal.value = 1
```

- Reading a Signal:

```
x = dut.signal.value
```

- Reading a Nested Signal:

```
y = dut.module.signal.value
```

- CocoTB always uses non-blocking assignment (<=)
  - Values will not get updated in your **DUT** until a timestep occurs
  - Reading a value reads the value from the last time step, not updated value

```
dut.signal.value = 0
await RisingEdge(dut.clk_w) # Timestep
dut.signal.value = 1
x = dut.signal.value
# x will be 0
```

Simulation Time Steps

Simulation only happens during a timestep. Each main SV time step has a CoCoTB analogue.

	System Verilog	CocoTB
Delay	`#(1)`	`await Timer(1, units="ns")`

Creating a Clock	`always begin`
	`  #(20)`
	`  clk <= ~clk;`
	`end`
		`clock = Clock(dut.clk_i, 10)`
		`cocotb.start_soon(clock.start())`

Clock Edge	`@(posedge clk);` `@(negedge clk);`	`await RisingEdge(dut.clk)` `await FallingEdge(dut.clk)`
Assertions	`assert (clk == 0);`	`assert clk == 0`

There is no version of system verilog's **`wait`**, other than a while loop with delays / edges.

CocoTB Concurrency

- CocoTB uses Python's asyncio library to handle concurrency
- Declare functions as **`async`** to use them as coroutines
  - Each can interact with DUT
  - Each can fail
- Use **`await`** to suspend a coroutine as it waits for another one
- Equivalent to having multiple **`always`** blocks in system verilog

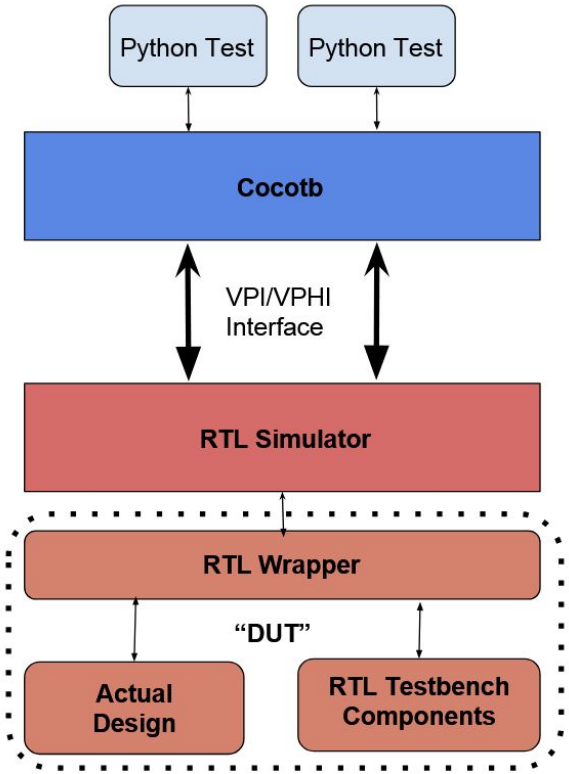
```
async def function1 (dut):
    pass
async def function2 (dut):
    pass

@cocotb.test()
async def main_test(dut):
    task1 = cocotb.start_soon(function1(dut))
    task2 = cocotb.start_soon(function2(dut))
    await task1
    await task2
```

CocoTB Tradeoffs

+ CocoTB enables use of a higher level language
  + Great for file IO, simulating external devices (VGA, Memories, etc.)
  + Libraries for common interfaces (AXI, Wishbone, etc.)

- Slower than raw SV simulation
  - Python is slow enough, moving back and forth between Python and Simulator is extra slow

- Can’t do top level integration
  - In SV, can instantiate more than one module in testbench and hook them up
  - Can’t instantiate more than one module in cocotb
  - If more modules are needed, have to create a verilog wrapper top level module



CocoTB Bus Extension

Fix Outdated CocoTB Fork, Case Sensitivity (#1)

fkwilken authored on Dec 10, 2023

- CocoTB allows powerful abstraction of buses
  - Wishbone, AXI, etc.
  - Verify your design against open source standards

- Can encapsulate signals in an object
  - Wishbone Example uses a dictionary to define signal names
  - Abstracted reads and writes

```
 # Attach Wishbone Master to Bus
wbs = WishboneMaster(dut, "wbs", dut.wb_clk_i,
                           width=32,   # size of data bus
                           timeout=10, # in clock cycle number
                           signals_dict={"cyc":  "cyc_i",
                                         "stb":  "stb_i",
                                         "we":   "we_i",
                                         "adr":  "adr_i",
                                         "datwr":"dat_i",
                                         "datrd":"dat_o",
                                         "ack":  "ack_o",
                                         "sel":  "sel_i" })

# Create Write Operation
wb_op = WBOp(adr,dat,0,sel=0xF)
wbs.send_cycle([wb_op])
# Create Read Operation
wb_op = WBOp(adr,None,0,sel=0xF)
read_resp = wbs.send_cycle([wb_op])
```

CocoTB Example

https://github.com/cocotb/cocotb/blob/master/examples/adder/tests/test_adder.py

https://github.com/cocotb/cocotb/blob/master/examples/simple_dff/test_dff.py

References

- https://docs.cocotb.org/en/stable/
- https://indico.cern.ch/event/776422/attachments/1769690/2874927/cocotb_talk.pdf
- https://www.chipverify.com/systemverilog/systemverilog-class