2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

CPAS: Cocotb Power Aware Simulation Framework

Ahmed Alsawi, AMD, Dublin, Ireland (*ahmed.alsawi@amd.com*)

Liam O'Reilly, AMD, Dublin, Ireland (*Liam.OReilly@amd.com*)

Evin Hughes, AMD, Dublin, Ireland (*evin.hughes@amd.com*)

***Abstract*—Power aware design and verification are important requirements of the modern system on chip flow. With the proliferation of free and open-source design and verification tools, power aware simulation is still only supported with commercial simulators. To enable power aware simulation with an open-source toolchain, a framework is introduced to enable design and verification engineers to specify power intent and run power aware simulation using Python and Cocotb. The framework implements power aware specifications with a Python frontend using popular and easy language to define and drive power aware intent. The framework also integrates with Yosys to extract design information and Cocotb to implement power aware semantics.**

Keywords—Cocotb, Open Source, Power Aware Simulation, Python, Yosys

I. Introduction

Power aware (PA) simulation is an essential part of the design and verification flow, as processing power and battery life are not only important but also conflicting requirements. The design and verification of low power system on chip (SOC) have gone through major improvements and innovations over the past few decades. As design density increases exponentially and technology dimensions get smaller, the power dissipation increases drastically, leading to cooling and reliability problems. For battery-powered devices, the requirement of adding more features and increasing the chip frequency results in increasing the power consumption which results in shorter battery lifetime. As a result, the power budget became one of the most important concerns along with cost, area, and timing.

A. *Power aware techniques*

Clock gating and multi-voltage techniques are the most popular techniques developed to address power consumption. Clock gating is turning off clocks when blocks are not in use. Multi-voltage design introduced multiple variants depending on power budget requirements:

- Static voltage scaling (SVS) uses different blocks operating on fixed different voltages depending on functionality and performance requirements.
- Dynamic voltage and frequency scaling (DVFS) enables each block to operate on a different voltage and frequency depending on load and performance requirements.

B. Unified power format

Unified power format (UPF) [1] is the de-facto standard to specify power intent. It is interoperable and supported by commercial tools. UPF specifies power aware semantics by defining power domains, ports, and nets. It also defines power blocks such as isolation, retention, and power switches. Beside PA simulations, UPF is used by synthesizers for power elements insertion based on technology-specific libraries.

C. *Cocotb Library*

Coroutine based co-simulation testbench (Cocotb) [2] is a concurrent coroutine library to enable writing testbenches for Verilog and VHDL designs in Python. Cocotb is simulator agnostic and supports both commercial and open-source simulators by implementing a C++/Python layer over native interfaces like Verilog programming interface (VPI). As Cocotb only provides access to design signals, there have been efforts to implement functional verification features on top of Cocotb. pyuvm [3] implements the universal verification methodology (UVM) class library allowing reusable verification components with Cocotb. In [4], authors implemented Cocotb-coverage to add coverage and randomization semantics such as features specified in Systemverilog language manual [5]. In [6], authors implemented a fault injection framework using Cocotb to evaluate fault injection policies.

D. Yosys Synthesizer

Yosys [7] is an open-source Verilog synthesizer that generates a generic or implementation netlist based on technology-specific libraries. Yosys provides a command line interface to run synthesis phases. Also, it provides Python bindings to enable native integration.

E. Cocotb Power Aware Simulation Framework

Currently, UPF is only supported with commercial simulators which leaves a gap in the open-source design and verification flow. In this work, Cocotb power aware simulation (CPAS) framework is developed to specify power intent and run PA simulations using Python. The framework implements a UPF frontend to process UPF commands and uses Yosys synthesizer to extract information required for PA simulations. The advantage of CPAS, it enables design and verification engineers to run PA simulations with free open-source toolchain (Python, Cocotb, iverilog, Yosys) enabling prototyping, development, and verification of power intent. The framework also enables integration with Python standard library and user packages enabling innovative techniques for processing UPF or debugging PA simulations.

II. CPAS IMPLEMENTATION

The CPAS framework leverages open-source tools Yosys and Cocotb to enable PA simulations with Python. The framework implements three engines: Yosys engine, power engine, and Cocotb engine shown in Figure 1. The framework takes the following inputs: Cocotb power test, Python power specifications and design files. And it generates text logs to trace the power nets, ports, domains, isolation, retention, and power switch elements.

The modular architecture enables Yosys and Cocotb engines to abstract the details of the underlying tools. The power engine encapsulates all power aware semantics. This enables easy development and modifications. For example, the power engine can be modified for a newer UPF version while keeping the other engines intact.

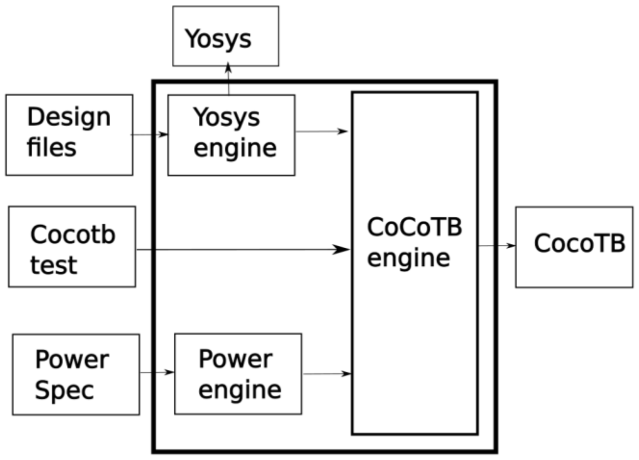

Figure 1 CPAS architecture.

A. Power Engine

The power engine parses Python-based power specifications and calculates domains state when power ports or power switch control signals change. It also triggers Cocotb engine to drive corruption, isolation, and retention semantics. The engine supports the following UPF 1.0 commands:

- set_design_top
- set_scope
- create_supply_net
- create_supply_port
- connect_supply_net
- create_power_switch
- create_power_domain
- set_domain_supply_net
- set_retention
- set_retention_control
- set_isolation
- set_isolation_control

Beside the power intent commands, there are two commands to control supply ports state:

- supply_on
- supply_off

B. *Yosys Engine*

Yosys engine is needed to extract important information required for domain corruption, retention, and isolation. The engine calls Yosys using the Python bindings to extract the following information:

- Synthesizable flip-flops
- Port direction: required as Cocotb information model does not support port direction.

C. *Cocotb Engine*

Cocotb engine is the core controller of CPAS framework as it coordinates with Yosys and power engines to monitor signals and use force and release to implement power aware semantics.

D. *Use Model*

CPAS can be integrated with a Cocotb test using the following steps:

1. Define CPAS Python power intent.
2. Create CPAS instance.
3. Use supply_on/supply_off to drive pad voltage changes or control signals to drive isolation, retention, and power switches.

The power intent method defines power domains, nets, ports, and other power elements using power engine APIs the same way TCL UPF commands are used.

```
def power_intent(pe):
        top_pd = pe.create_power_domain("top_pd", include_scope=True)
        pe.create_supply_port("VDD_PORT", top_pd)
        pe.create_supply_net("VDD", top_pd)
        pe.set_domain_supply_net(top_pd, top_pd.VDD, top_pd.VSS)
        pe.connect_supply_net(top_pd.VDD, [top_pd.VDD_PORT])
```

Then, CPAS initialization interface requires design top, testbench top, list of design RTL and the power intent method defined above.

```
cpas = CPAS( design_top = top.dut, top = top, rtl_source = verilog_sources, power_intent = power_intent)
```

The power test can drive the control signals, or call supply_on/supply_off to change any port state and trigger power aware semantic.

E. *Possible Applications*

As CPAS uses Cocotb as an execution environment, a typical use case is enabling PA simulations with a verification environment already using Cocotb. Another use case is creating a small CPAS testcase for prototyping and experimenting with power topology and power controller logic. Also, In the future, CPAS could work as a backend to generate UPF files from specifications, graphical user interface or translated from another format such as Excel sheets, or JSON.

III. Results

The results show CPAS examples including test stimulus, generated messages, and waveform for domain corruption, retention, isolation, and power switch scenarios. The block diagram in Figure 2 shows simple design including three blocks (U1, U2, and U3) and power aware elements used to implement the power intent.

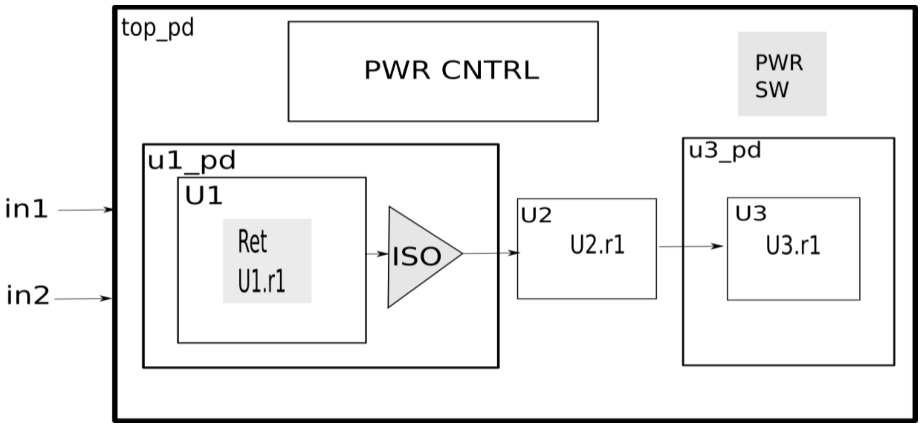

Figure 2 Sample power design.

The waveform in Figure 3 shows signal values of ports and flip-flops with markers at the start and end of power scenarios.

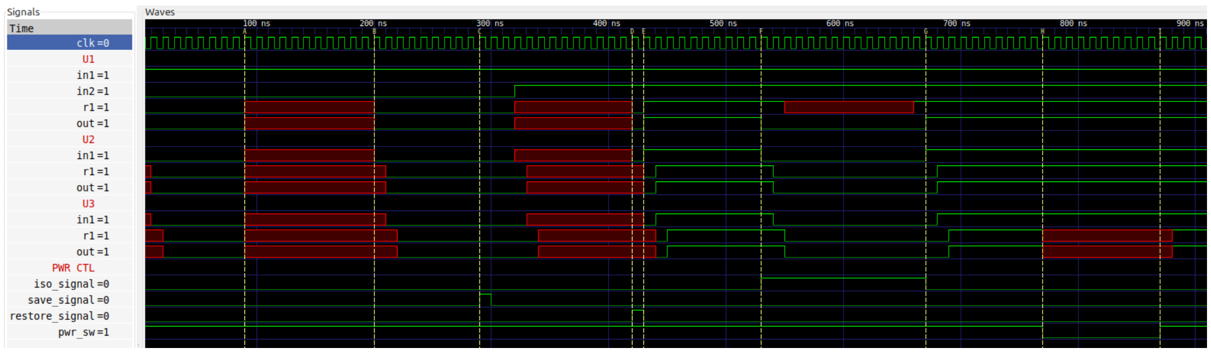

Figure 3 Power aware scenarios waveform.

A. Power Domain

Domain top_pd is created by create_power_domain and supply nets are connected in top_pd domain with set_domain_supply_net.

```
pe.set_design_top("top/dut")
pe.set_scope("")

top_pd = pe.create_power_domain("top_pd", include_scope=True)

pe.create_supply_port("VDD_PORT", top_pd)
pe.create_supply_port("VSS_PORT", top_pd)
pe.create_supply_net("VDD", top_pd)
pe.create_supply_net("VSS", top_pd)

pe.set_domain_supply_net(top_pd, top_pd.VDD, top_pd.VSS)

pe.connect_supply_net(top_pd.VDD, [top_pd.VDD_PORT])
pe.connect_supply_net(top_pd.VSS, [top_pd.VSS_PORT])
```

To trigger domain corruption, Cocotb test calls supply_off on top_pd/VDD_PORT to turn off the power port. The power engine recalculates top_pd state and other domains directly and indirectly connected to port VDD_PORT.

```
cpas.power_engine.supply_on("top_pd/VDD_PORT", 1)
cpas.power_engine.supply_on("top_pd/VSS_PORT", 1)
...
cpas.power_engine.supply_off("top_pd/VDD_PORT")
```

During simulation, CPAS prints messages to log power events such as domain state change, corruption, retention, isolation, and power switch activity. Messages are printed when domain u1_pd changes from normal to corrupt after VDD goes to off. As domain u1_pd goes to corrupt, it is expected that flip-flops and connected nets under u1_pd will be corrupted. Figure 3, between markers A and B, shows flip-flop r1 and connected net corruption inside U1 instance. Note that top_pd and u3_pd also change to corrupt state as they are connected to top_pd/VDD_PORT.

```
PAEngine: Network update: top_pd.VDD_PORT --> top_pd.VDD_PORT to (<POWERSTATE.OFF: 2>, 0)
PAEngine: Network update: top_pd.VDD --> u1_pd.VDD_PORT to (<POWERSTATE.OFF: 2>, 0)
PAEngine: Network update: u1_pd.VDD_PORT --> u1_pd.VDD to (<POWERSTATE.OFF: 2>, 0)
PAEngine: Update Domain top_pd to vdd: (<POWERSTATE.OFF: 2>, 0) gnd:(<POWERSTATE.FULL_ON: 1>, 0) simstate
SIMSTATE.CORRUPT
Cocotb:  corrupt domain top_pd with scope top.dut signals: [ModifiableObject(top.dut.r1)]
```

B. *Retention*

For retention scenario, set_retention and set_retention_control methods are used to define retention policy and control signals for domain u1_pd.

```
ret = pe.set_retention(
  name="u1_ret",
  domain=u1_pd,
  retention_power_net=top_pd.VDD
)

pe.set_retention_control(ret,
  domain=u1_pd,
  save_signal=("m_pwr_ctl/save_signal", power_engine.RETTYPE.posedge),
  restore_signal=("m_pwr_ctl/restore_signal", power_engine.RETTYPE.posedge),
)
```

To trigger retention for flip-flops in domain u1_pd, Cocotb test drives positive edge on save and restore signals to control retention during domain u1_pd corruption.

```
# retain value
top.dut.m_pwr_ctl.save_signal.value = 1
...
top.dut.m_pwr_ctl.save_signal.value = 0

cpas.power_engine.supply_off("u1_pd/VDD_PORT")
# During retention
cpas.power_engine.supply_on("u1_pd/VDD_PORT",1)

# restore old value
top.dut.m_pwr_ctl.restore_signal.value = 1
...
top.dut.m_pwr_ctl.restore_signal.value = 0
```

Cocotb engine prints the following messages to show U1.r1 is saved and restored according to the retention controller signals. Figure 3, between markers C and D, shows flip-flop U1.r1 corrupted on domain collapse then restored back to 0. After one clock cycle, it gets assigned to 1 during the normal operation of the flip-flop at marker E.

```
Cocotb: RET Save domain u1_pd with scope top.dut.U1 signals: [ModifiableObject(top.dut.U1.r1)]

PAEngine: Update Domain u1_pd to vdd: (<POWERSTATE.OFF: 2>, 0) gnd:(<POWERSTATE.FULL_ON: 1>, 1) simstate
SIMSTATE.CORRUPT

PAEngine: Update Domain u1_pd to vdd: (<POWERSTATE.FULL_ON: 1>, 1) gnd:(<POWERSTATE.FULL_ON: 1>, 1) simstate
SIMSTATE.NORMAL

Cocotb: RET RESTORE domain u1_pd with scope top.dut.U1 signals: [ModifiableObject(top.dut.U1.r1)]
```

C. Isolation

For isolation scenario, set_isolation and set_isolation_control methods are used to create isolation element and connect isolation control signal.

```
iso = pe.set_isolation(name="u1_iso",
 domain=u1_pd,
 isolation_power_net=top_pd.VDD,
 clamp_value=power_engine.CLAMPVALUE.V0)

pe.set_isolation_control(iso,
 domain=u1_pd,
 isolation_signal="m_pwr_ctl/iso_signal",
 isolation_sense=power_engine.ISOSENSE.HIGH)
```

To control isolation, Cocotb test asserts the isolation signal by changing iso_signal before calling supply_off and de-asserts iso_signal after the supply is back to on again.

```
top.dut.m_pwr_ctl.iso_signal.value = 1
cpas.power_engine.supply_off("u1_pd/VDD_PORT")
...
cpas.power_engine.supply_on("u1_pd/VDD_PORT",1)
top.dut.m_pwr_ctl.iso_signal.value = 0
```

Cocotb engine prints messages for the isolation of output port U1.out before turning off u1_pd. The isolation is shown in Figure 3 between marker F and G, where U1.out output port is isolated to 0 stopping x-propagation from U1.r1 to U1.out port.

```
Cocotb: ISO domain u1_pd with scope top.dut.U1 signals: [ModifiableObject(top.dut.U1.out)] set

WARNING PAEngine: Update Domain u1_pd to vdd: (<POWERSTATE.OFF: 2>, 0) gnd:(<POWERSTATE.FULL_ON: 1>, 1)
simstate SIMSTATE.CORRUPT

PAEngine: Update Domain u1_pd to vdd: (<POWERSTATE.FULL_ON: 1>, 1) gnd:(<POWERSTATE.FULL_ON: 1>, 1) simstate
SIMSTATE.NORMAL

Cocotb DEISO domain u1_pd with scope top.dut.U1 signals: [ModifiableObject(top.dut.U1.out)]
```

D. Power switch

For power switch scenario, create_power_switch is used to define power switch in top_pd domain driving supply net u3_vdd from power port top_pd.VDD.

```
sw = pe.create_power_switch(
   name="u1_pwr_sw",
   domain=top_pd,
   output_supply_port=u3_vdd,
   input_supply_port=[
      ("my_sw_input_port", top_pd.VDD)
   ],
   control_port=[
      ("my_sw_control_port","m_pwr_ctl/pwr_sw")
   ],)
```

Cocotb test can de-assert and assert pwr_sw control signal to turn on and off the power switch which affects u3_pd domain.

```
top.dut.m_pwr_ctl.pwr_sw.value = 0
...
top.dut.m_pwr_ctl.pwr_sw.value = 1
```

Cocotb engine prints messages about domain u3_pd changing to corrupt and back to normal again when the switch control signal is de-asserted and asserted. Between markers H and I in Figure 3, The waveform shows the corruption of u3_pd domain while the switch is turned off.

```
PAEngine: Updating SWITCH State  to (<POWERSTATE.OFF: 2>, 0) output port  Name: VDD -- Type: SupplyNet -- conn:
VDD_PORT

PAEngine: Update Domain u3_pd to vdd: (<POWERSTATE.OFF: 2>, 0) gnd:(<POWERSTATE.FULL_ON: 1>, 1) simstate
SIMSTATE.CORRUPT

PAEngine: Updating SWITCH State  to (<POWERSTATE.FULL_ON: 1>, 1) output port  Name: VDD -- Type: SupplyNet --
conn: VDD_PORT

PAEngine: Update Domain u3_pd to vdd: (<POWERSTATE.FULL_ON: 1>, 1) gnd:(<POWERSTATE.FULL_ON: 1>, 1) simstate
SIMSTATE.NORMAL
```

IV. Future improvements

The main limitation of CPAS that it currently supports UPF 1.0 commands. That said, due to the modular pythonic architecture, new versions can be supported in the power engine keeping other engines unchanged. As CPAS provides a framework to describe UPF semantics in Python, UPF files can be parsed or generated from CPAS Python specifications. Also, it is possible to generate CPAS specification from graphical user interface. As a result, the following future improvements are planned:

- Implement newest UPF commands.
- Implement graphical user interface to generate Python CPAS specification.
- Implement UPF-to-CPAS parser.
- Implement CPAS-to-UPF generator.
- Add power aware instrumentation for VCD logging and visualization of power events.
- Support UPF level shifters.

V. Conclusion

Power aware design and verification are essential tasks of the modern SOC design flow. That said, UPF is the industry standard for PA simulation and only supported by commercial simulators leaving a gap in open-source design and verification flow.

The proposed CPAS framework implements power intent specifications using Python and integrates with Cocotb to create power aware tests and verify power controller logic. The results show power aware specifications for corruption, isolation, retention, and power switch. The results also show messages and waveform for typical power aware scenarios.

CPAS enables integration of PA simulation into Cocotb tests or prototyping power design purely in Python. As CPAS is developed in Python, it enables access to the Python standard library and user-defined packages for advanced processing and visualization.

References

[1] ”IEEE Standard for Design and Verification of Low Power Integrated Circuits,” in IEEE Std 1801-2009 , vol., no., pp.1-218, 27 March 2009, doi: 10.1109/IEEESTD.2009.4809845.

[2] ”coroutine based cosimulation library” https://github.com/Cocotb/Cocotb (accessed April 20, 2023).

[3] ”Universal Verification Methodology implemented in Python instead of Systemverilog” https://github.com/pyuvm/pyuvm (accessed April 20, 2023).

[4] ”Functional Coverage and Constrained Randomization Extensions for Cocotb” https://github.com/mciepluc/Cocotb-coverage (accessed April 20, 2023).

[5] ”IEEE Standard for Systemverilog–Unified Hardware Design, Specification, and Verification Language,” in IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012) , vol., no., pp.1-1315, 22 Feb. 2018, doi: 10.1109/IEEESTD.2018.8299595.

[6] ”Cocotb Fault Injection” https://gitlab.cern.ch/tmrg/Cocotb fault injection (accessed April 20, 2023).

[7] ”Yosys Open SYnthesis Suite.” https://github.com/YosysHQ/yosys (accessed April 20, 2023).