Open Source RTL Playing

Starting with an async_fifo design verification task

Yilou Wang

How to model async_fifo, RTL or SystemC?

RTL or SystemC?

- SystemC: Transaction Level Modeling
- RTL: Signal Level

A toy project in SystemC

https://github.com/YilouWang/Toy4Joy_SystemC

Section1: FiFo Modeling and Testing

1. modeling 4 fifos

   (signal-level-normal fifo,

   signal-level-FWFT(first-word-fall-through) fifo,

   transaction-level fifo with sc_interface,

   transaction-level fifo with TLM)

2. comparing the performance of 4 fifos

Result and Conclusion

Test0: Fifos have a size of 16, and 8 to 24 bytes are written to/read from them randomly.

Test1: Fifos have a size of 2048, and 512 to 1536 bytes are written to/read from them randomly.

Conclusion: TLM — Increased simulation speed and Reduced modeling effort

Async FiFo

Async: two different clks, w_clk and r_clk

Main problem: metastability, which shows in signal level

So, I choose to model async_fifo in signal level (RTL).

RTL Playing

https://github.com/YilouWang/RTL_Playing/tree/main/Asyn_Fifo/DUT

I model async_fifo in SystemVerilog and run a simple simulation in Verilator with a simple C++ testbench to drive signals.

https://github.com/YilouWang/RTL_Playing/tree/main/Asyn_Fifo/py_tb

After modeling the async_fifo, I use the open source Pyuvm-cocotb to build the verification environment to do the verification task.

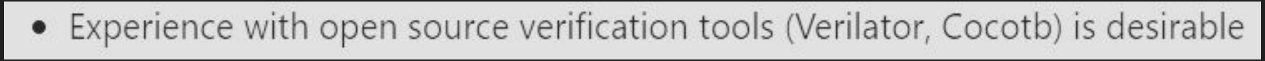
Why Verilator?

·Open Source

·Fast Simulation Speed

·First-time-exploration (This is my first time using verilator, and it's a great opportunity to learn more about it.)

·Qualification in job description



- Experience with open source verification tools (Verilator, Cocotb) is desirable

Why PyUVM-Cocotb?

Open Source

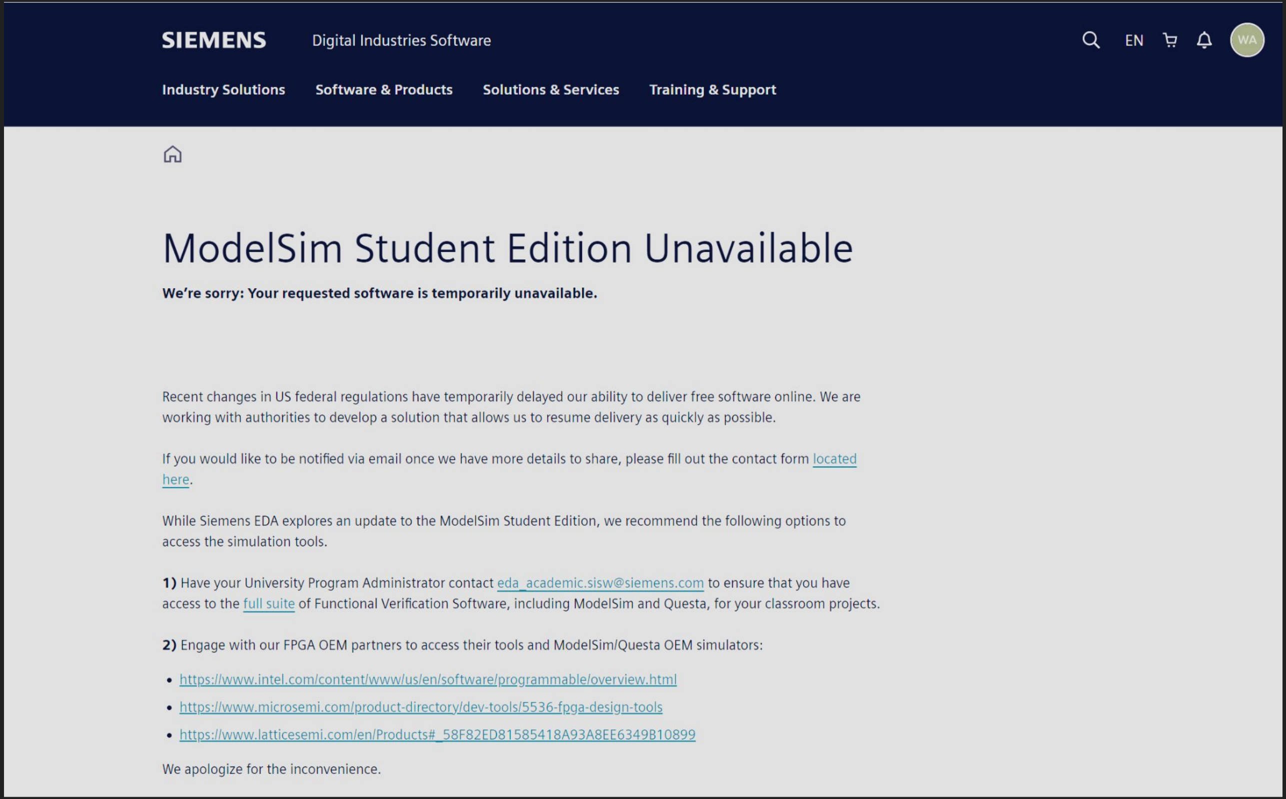
Easy Python programming

Supported by Verilator

Already familiar with it (experience in my master thesis)

sv-uvm is not fully supported in Verilator and Modelsim is unavailable.

Modelsim unavailable



ModelSim Student Edition Unavailable

We're sorry: Your requested software is temporarily unavailable.

Recent changes in US federal regulations have temporarily delayed our ability to deliver free software online. We are working with authorities to develop a solution that allows us to resume delivery as quickly as possible.

If you would like to be notified via email once we have more details to share, please fill out the contact form located here.

While Siemens EDA explores an update to the ModelSim Student Edition, we recommend the following options to access the simulation tools.

1) Have your University Program Administrator contact eda_academic.sisw@siemens.com to ensure that you have access to the full suite of Functional Verification Software, including ModelSim and Questa, for your classroom projects.

2) Engage with our FPGA OEM partners to access their tools and ModelSim/Questa OEM simulators:

- https://www.intel.com/content/www/us/en/software/programmable/overview.html
- https://www.microsemi.com/product-directory/dev-tools/5536-fpga-design-tools
- https://www.latticesemi.com/en/Products#_58F82ED81585418A93A8EE6349B10899

We apologize for the inconvenience.

RTL Playing

https://github.com/YilouWang/RTL_Playing/tree/main/Asyn_Fifo/DUT

I model async_fifo in SystemVerilog and run a simple simulation in Verilator with a simple C++ testbench to drive signals.

Async FiFo

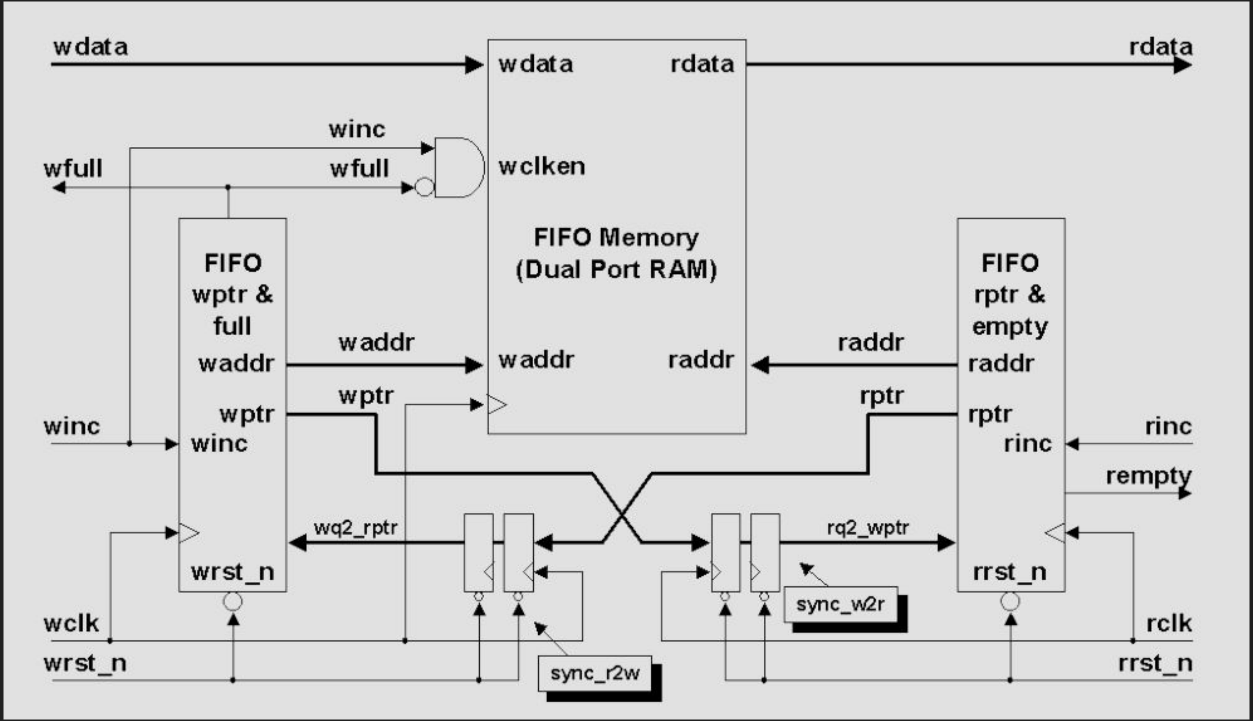


Figure cited from http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

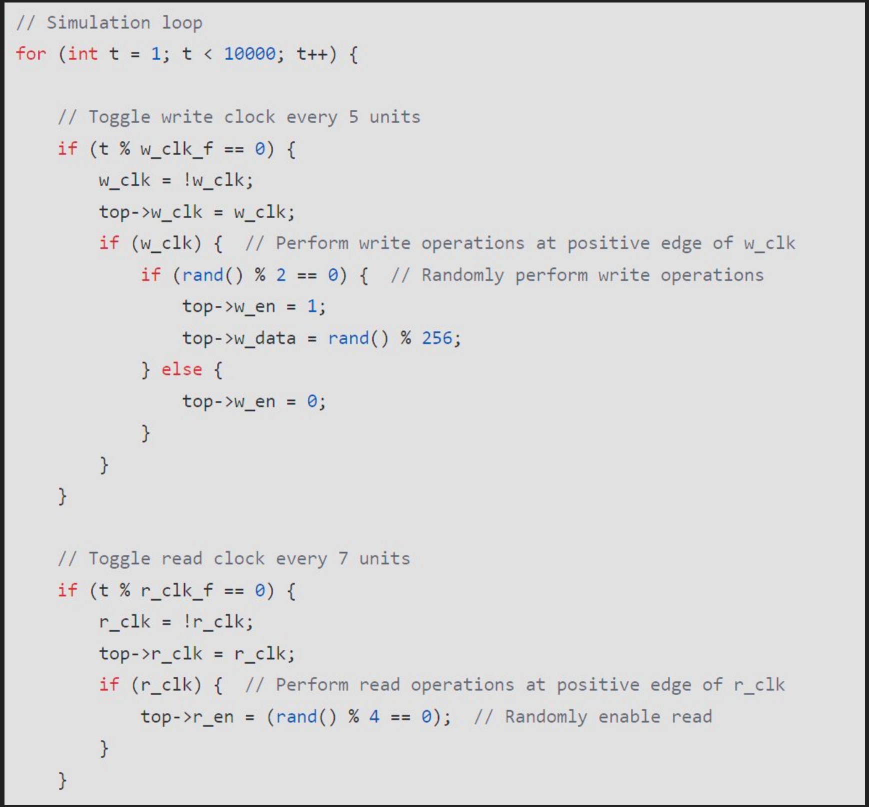
Async FiFo

sync_2ff.sv: a synchronizer that uses double flip-flops to greatly reduce the probability of metastability.

empty/full_checker.sv: determines if the fifo is full or empty and converts the write/read pointer into Gray code.

fifo_mem.sv: stores fifo data

simple_testbench.cpp



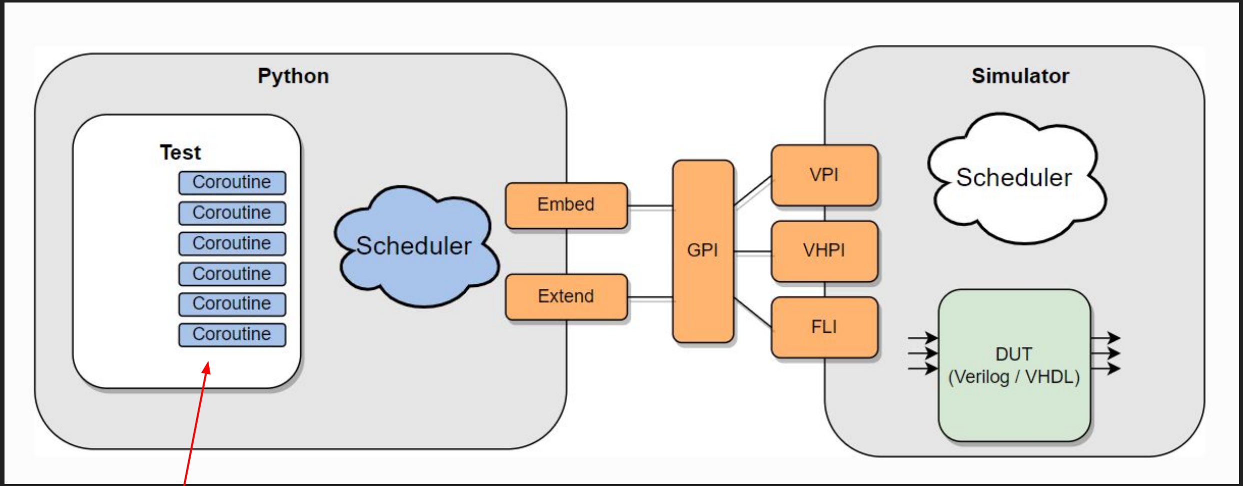
```cpp
// Simulation loop
for (int t = 1; t < 10000; t++) {

    // Toggle write clock every 5 units
    if (t % w_clk_f == 0) {
        w_clk = !w_clk;
        top->w_clk = w_clk;
        if (w_clk) {  // Perform write operations at positive edge of w_clk
            if (rand() % 2 == 0) {  // Randomly perform write operations
                top->w_en = 1;
                top->w_data = rand() % 256;
            } else {
                top->w_en = 0;
            }
        }
    }

    // Toggle read clock every 7 units
    if (t % r_clk_f == 0) {
        r_clk = !r_clk;
        top->r_clk = r_clk;
        if (r_clk) {  // Perform read operations at positive edge of r_clk
            top->r_en = (rand() % 4 == 0);  // Randomly enable read
        }
    }
```

RTL Playing

https://github.com/YilouWang/RTL_Playing/tree/main/Asyn_Fifo/py_tb

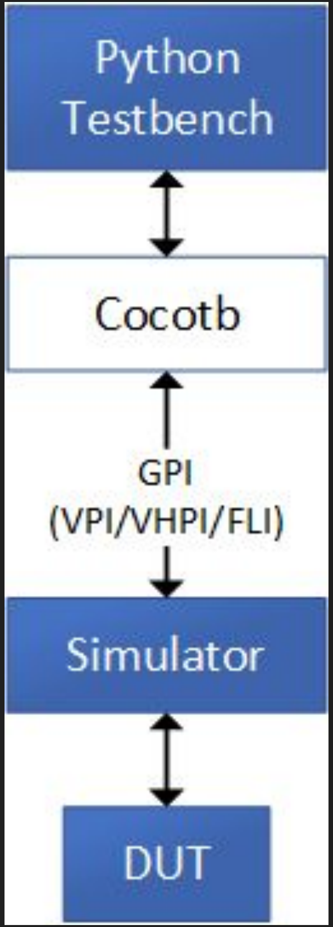
After modeling the async_fifo, I use the open source Pyuvm-cocotb to build the verification environment to do the verification task.

pyuvm-cocotb



cocotb.trigger()

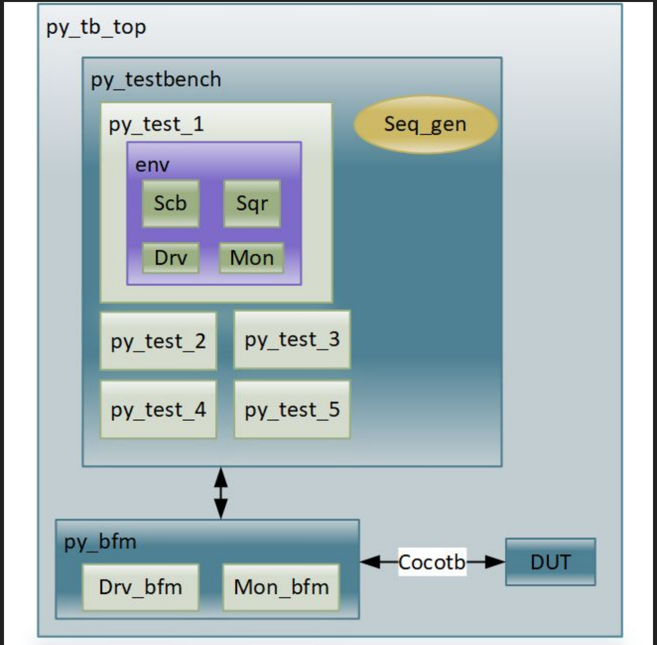
Figure cited from cocotb manual document



py_tb

testbench.py: uvm components and uvm framework

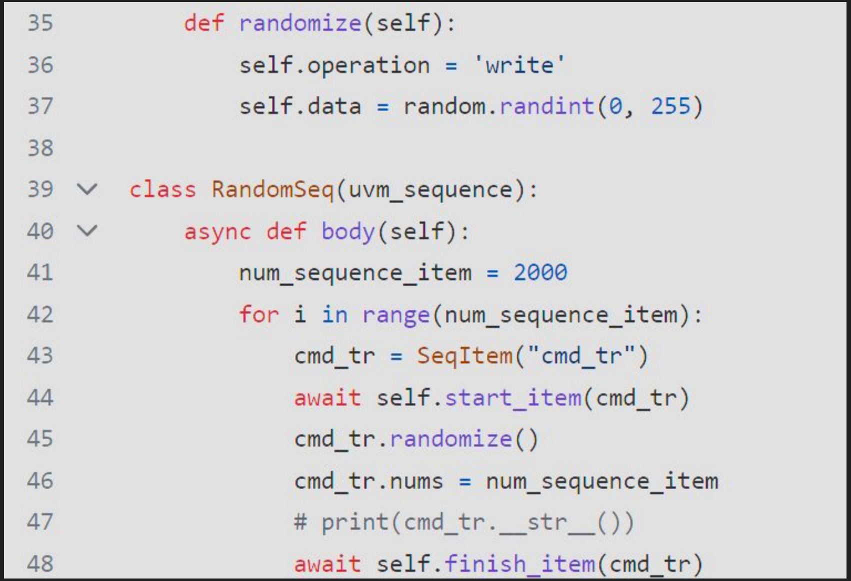
async_fifo_utils.py: BFM for testbench to communicates with DUT



Random_Sequence

Generating random w_data

Sequencer sends sequence to driver



```
    def randomize(self):
        self.operation = 'write'
        self.data = random.randint(0, 255)

class RandomSeq(uvm_sequence):
    async def body(self):
        num_sequence_item = 2000
        for i in range(num_sequence_item):
            cmd_tr = SeqItem("cmd_tr")
            await self.start_item(cmd_tr)
            cmd_tr.randomize()
            cmd_tr.nums = num_sequence_item
            # print(cmd_tr.__str__())
            await self.finish_item(cmd_tr)
```

Driver

Initiates a fork … join_any, one for pushing data, the other for popping data

```python
async def run_phase(self):
    await self.launch_tb()
    push_task = cocotb.start_soon(self.push_data())
    pop_task = cocotb.start_soon(self.pop_data())
    await First(Join(push_task), Join(pop_task)) # fork ... join_any
```

Pushing and popping data with a random-generated delay

```python
async def push_data(self):
    while True:
        seq_item = await self.seq_item_port.get_next_item()
        self.count += 1
        w_delay = random.randint(0, self.w_delay_max)
        # print(seq_item.nums, self.count)
        self.logger.info(f"Push: w_data,{seq_item.data} into fifo with a delay, {w_delay}")
        await self.bfm.push_with_delay(seq_item.data, seq_item.operation, w_delay)
        self.seq_item_port.item_done()
        if self.count >= seq_item.nums:
            await self.bfm.wait_r_clk_cycles(50)
            self.drive_task_finished.set()
            print("notified")
```

monitor & scoreboard

cmd_monitor, monitors w_data, and sends to scoreboard as the expected result

result_monitor, monitors r_data, and sends to scoreboard as the actual result

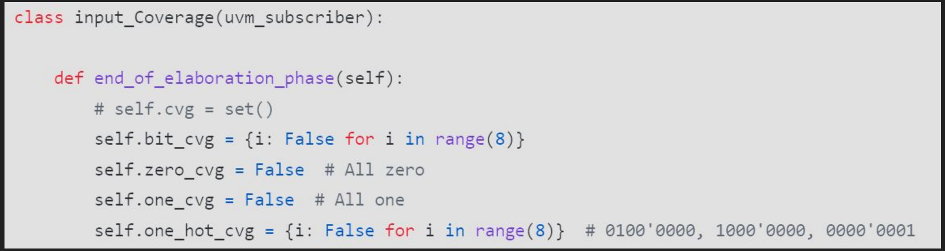
In scoreboard, using a fifo(uvm_tlm_analysis_fifo) to receive the data from two monitor, so…

No need to build an extra reference model in this case.

Coverage

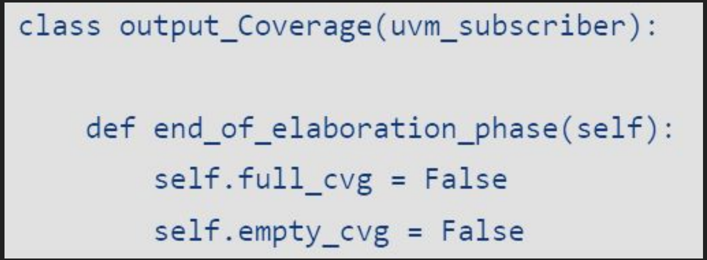
Two coverage classes in this case:

- input_coverage: w_data



```
class input_Coverage(uvm_subscriber):

    def end_of_elaboration_phase(self):
        # self.cvg = set()
        self.bit_cvg = {i: False for i in range(8)}
        self.zero_cvg = False  # All zero
        self.one_cvg = False  # All one
        self.one_hot_cvg = {i: False for i in range(8)}  # 0100'0000, 1000'0000, 0000'0001
```

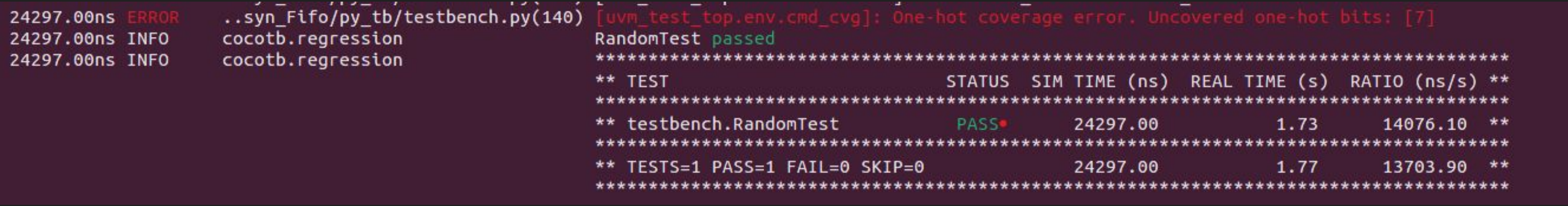
- output_coverage: w_full, r_empty



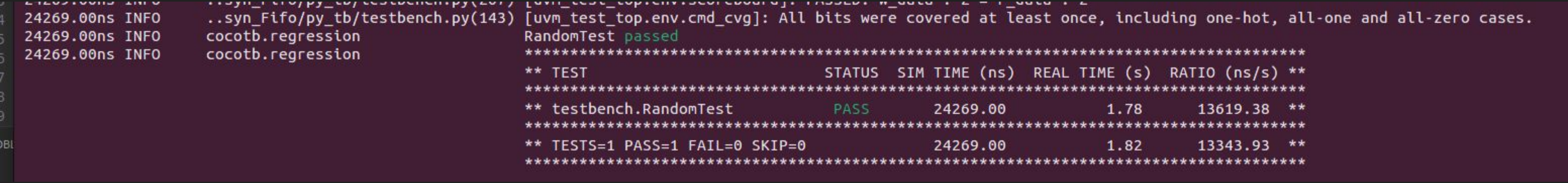
```
class output_Coverage(uvm_subscriber):

    def end_of_elaboration_phase(self):
        self.full_cvg = False
        self.empty_cvg = False
```

Results

nums = 2000, w_delay_max = 2, r_delay_max = 5



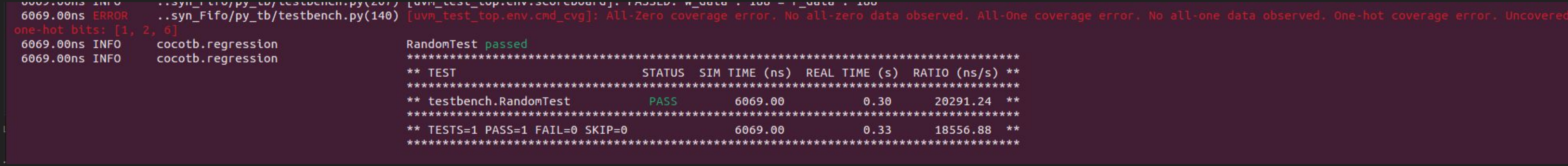
```
24297.00ns ERROR    ..syn_Fifo/py_tb/testbench.py(140) [uvm_test_top.env.cmd_cvg]: One-hot coverage error. Uncovered one-hot bits: [7]
24297.00ns INFO     cocotb.regression                  RandomTest passed
24297.00ns INFO     cocotb.regression                  ******************************************************************************
                                                       ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                       ******************************************************************************
                                                       ** testbench.RandomTest           PASS       24297.00           1.73      14076.10  **
                                                       ******************************************************************************
                                                       ** TESTS=1 PASS=1 FAIL=0 SKIP=0              24297.00           1.77      13703.90  **
                                                       ******************************************************************************
```



```
24269.00ns INFO     ..syn_Fifo/py_tb/testbench.py(143) [uvm_test_top.env.cmd_cvg]: All bits were covered at least once, including one-hot, all-one and all-zero cases.
24269.00ns INFO     cocotb.regression                  RandomTest passed
24269.00ns INFO     cocotb.regression                  ******************************************************************************
                                                       ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                       ******************************************************************************
                                                       ** testbench.RandomTest           PASS       24269.00           1.78      13619.38  **
                                                       ******************************************************************************
                                                       ** TESTS=1 PASS=1 FAIL=0 SKIP=0              24269.00           1.82      13343.93  **
                                                       ******************************************************************************
```

Results

nums = 200, w_delay_max = 5, r_delay_max = 2



```
 6069.00ns INFO     ..syn_Fifo/py_tb/testbench.py(207) [uvm_test_top.env.scoreboard]: PASSED: w_data : 188 = r_data : 188
 6069.00ns ERROR    ..syn_Fifo/py_tb/testbench.py(140) [uvm_test_top.env.cmd_cvg]: All-Zero coverage error. No all-zero data observed. All-One coverage error. No all-one data observed. One-hot coverage error. Uncovered
one-hot bits: [1, 2, 6]
 6069.00ns INFO     cocotb.regression                  RandomTest passed
 6069.00ns INFO     cocotb.regression                  **************************************************************************************
                                                       ** TEST                             STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                       **************************************************************************************
                                                       ** testbench.RandomTest              PASS        6069.00           0.30      20291.24  **
                                                       **************************************************************************************
                                                       ** TESTS=1 PASS=1 FAIL=0 SKIP=0                  6069.00           0.33      18556.88  **
                                                       **************************************************************************************
```

Conclusion

https://github.com/YilouWang/RTL_Playing

Modeling an async_fifo in SystemVerilog and Checking the basic functionality through a simple C++ testbench.

Architecting a verification environment in pyuvm, and verifying the async_fifo.

Future exploration

Create additional sequences to specifically test unique scenarios such as continuous writing, continuous reading, writing until full, and reading until empty.

Future exploration

Verilator can support partially UVM, recently presented by Krzysztof Bieganski on ORConf 2023 in Munich.

Video is from https://www.youtube.com/watch?v=2zOmpArtdH4

A simple memory uvm test has been contributed and it demonstrates the successful running with uvm in Verilator.

```
git clone https://github.com/antmicro/verilator-verification-features-tests.git
```

Therefore, it would be nice to put more effort in building a SV-UVM testbench running with Verilator and later integrate my thesis topic (reuse of SV-UVC in cocotb/pyuvm) into it.