

UVM Test Bench Top Level

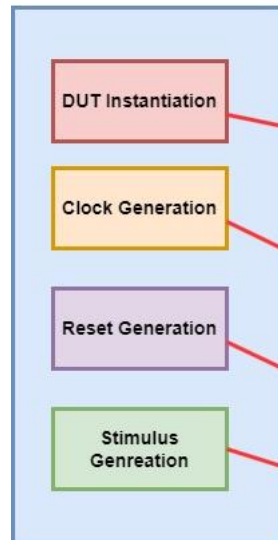
SyoSil 2025

Contents

- Top Level Example
 - SystemVerilog (SV)
 - CocoTB example
- Challenges with PyUVM
- UVM with conventional SV Interface
- Python interface object
- Running a test
- Phase objection
- Ending Test

SV-TB Top Level Example

Signals

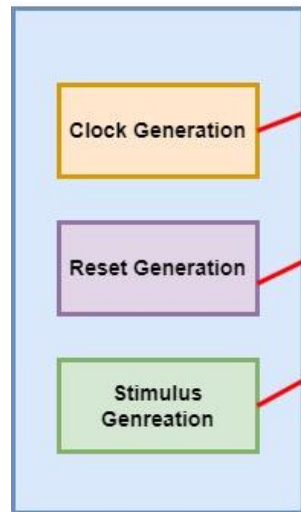


```
10 reg i_wr ;
11 reg [BUS_WIDTH-1] i_data;
12 // outputs as wires
13 wire [BUS_WIDTH-1] o_data ;
14 wire o_empty;
15 wire o_full ;
16
17 // Desin instantiation
18 fifo #(
19     .FIFO_DEPTH(FIFO_DEPTH),
20     .BUS_WIDTH (BUS_WIDTH )
21 ) DUT (
22     .clk (clk ), // Clock
23     .rst_n (rst_n ), // Asynchronous reset active low
24     .i_rd (i_rd ), // 1: rd
25     .i_wr (i_wr ), // 1 : wr
26     .i_data (i_data ), // input data
27     .o_data (o_data ), // output data
28     .o_empty(o_empty), // 1 : if fifo is empty
29     .o_full (o_full ) // 1 : if fifo is full
30 );
31 // generate clock
32 initial clk = 1'b1;
33 always #(CLK_PRD/2) clk = ~ clk;
34
35 initial begin
36     sys_rst();
37     single_rw();
38     repeat(5)
39         @(posedge clk);
40     $finish;
41 end
42 endmodule // tb_fifo
43
```

```
1 // reset test -- deassert reser for 5cc
2 task sys_rst();
3     rst_n <= 1'b0 ;
4     repeat(5)
5         @(posedge clk);
6     rst_n <= 1'b1 ;
7 endtask : sys_rst
8
9 task single_rw();
10     @(posedge clk)
11
12     if (!o_full) begin
13         i_data <= $random();
14         i_wr <= '1;
15         i_rd <= '0;
16     end
17     wait(!o_empty) begin
18         i_rd <= '1;
19         @(posedge clk)
20         $display("data read from fifo is %d",o_data);
21     end
22     if (o_data == i_data) begin
23         $display("***** Test Passs ***** Expected %d: Actual");
24     end else begin
25         $display("***** Test Fail ***** Expected %d: Actual");
26     end
27 endtask : single_rw
```

CocoTB Top Level Example

- No DUT Instantiation
- Handled by Icarus
- CocoTB gives "dut" object



```
""" FIFO Random Test """
@cocotb.test()
async def fifo_rand_test(dut):
    # generate clock
    clock = Clock(dut.clk, 10, units="us") # Create a 10ns period clock
    cocotb.start_soon(clock.start()) # Start the clock

    """ Reset DUT """
    await sys_rst(dut)
    """ Write item to FIFO """
    for x in range(10):
        item = random.randint(0,100);
        fifo_model('push',item)
        await write_item(dut, item)
    """ read item from FIFO """
    for y in range(10):
        actual_item = await read_item(dut)
        expctd_item = fifo_model('pop')
        print(" fifo result is",actual_item, "expected", expctd_item)
        assert actual_item == expctd_item , print(" *** MISMATCH ***")

    """ function to deassert reset """
    async def sys_rst(dut):
        print("Deassert Reset !!!")
        dut.rst_n.value = 0
        for i in range(5):
            await RisingEdge(dut.clk)
        dut.rst_n.value = 1

    """ function to write item to fifo """
    async def write_item(dut,item):
        if not dut.o_full.value :
            dut.i_rd.value = 0
            dut.i_wr.value = 1
            dut.i_data.value = item
            await RisingEdge(dut.clk)
        else:
            print(" fifo is full !!!")

    """ function to read item from fifo """
    async def read_item(dut):
        dut.i_wr.value = 0
        if not dut.o_empty.value :
            if not dut.i_rd.value:
                await FallingEdge(dut.clk)
                dut.i_rd.value = 1
                await RisingEdge(dut.clk)
                await RisingEdge(dut.clk)
                item = dut.o_data.value
                print("Data read from fifo is",item)
                return (item)
            else:
                await RisingEdge(dut.clk)
                item = dut.o_data.value
                print("Data read from fifo is",item)
                # await RisingEdge(dut.clk)
                return (item)
        else:
            print("fifo is empty !!!")
```

Challenges with Directed Testbenches

- **Scalability**

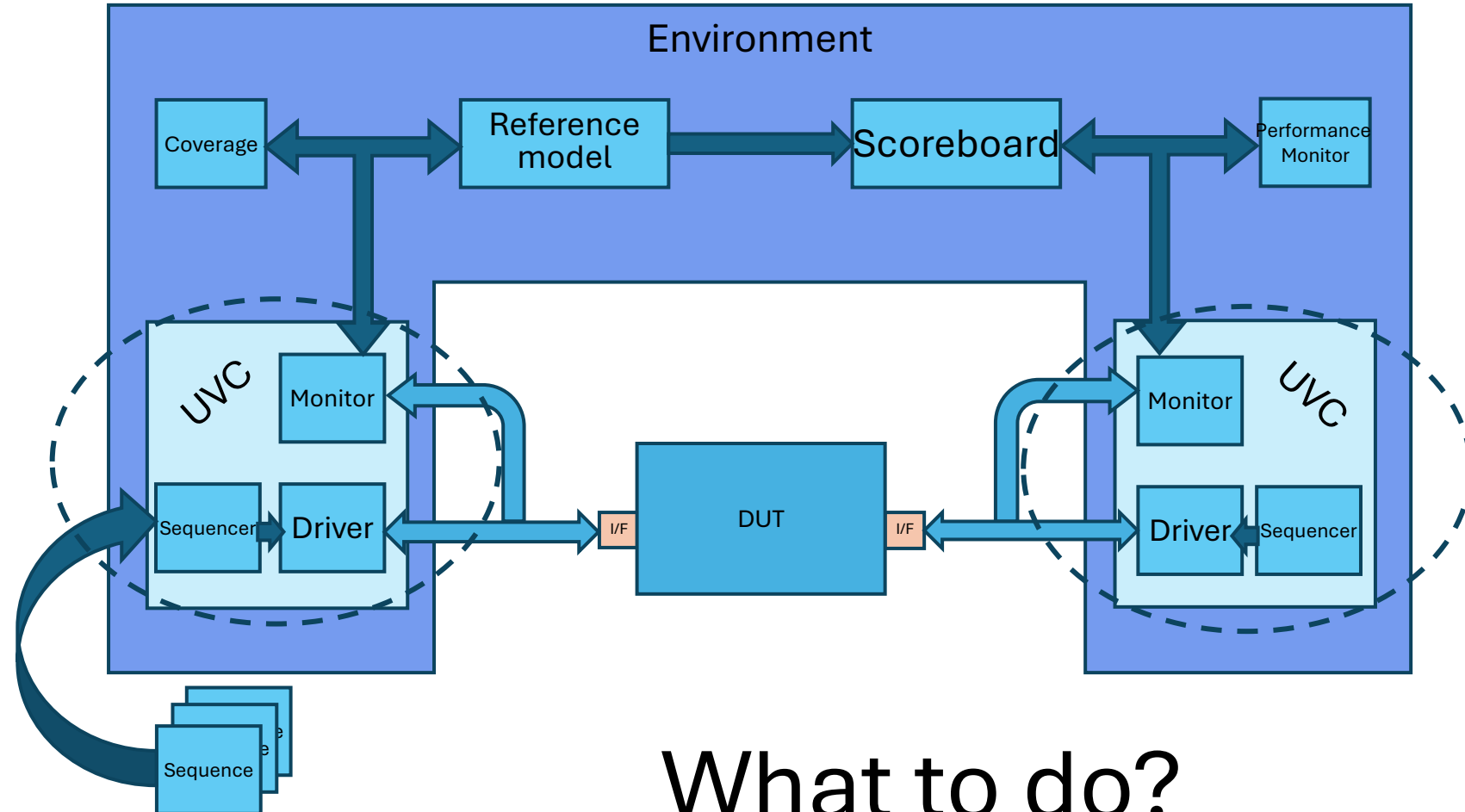
- Use CRV and CDV, write less tests
- But have to write coverage model

- **Reusability**

- Use UVCs
- Interfaces
 - Supported in PyUVM ?
 - NO! What can we do then?

UVM with conventional SV interface

```
interface fifo_intf(input logic clk,rst);  
  logic [31:0]data_in;  
  logic wr_en,rd_en;  
  logic [31:0]data_op;  
  logic full,empty;  
  
  clocking driver_cb@(posedge clk);  
    default input #1 output #1;  
    output data_in;  
    output wr_en,rd_en;  
    input data_op;  
    input full,empty;  
  endclocking  
  
  clocking monitor_cb@(posedge clk);  
    default input #1 output #1;  
    input data_in;  
    input wr_en,rd_en;  
    input data_op;  
    input full,empty;  
  endclocking  
  
  modport DRIVER(clocking driver_cb,input clk,rst);  
  modport MONITOR(clocking monitor_cb,input clk,rst);  
  
endinterface
```



What to do?

PyUVM with Interface Object

```
class fifo_intf:
    def __init__(self):
        self._data      = None
        self._valid     = None
        self._wr_en     = None
        self._rd_en     = None
        self._o_empty   = None
        self._o_full    = None

    def connect(self, _data, _valid, _wr_en, _rd_en, _o_empty, _o_full):
        self._data      = _data
        self._valid     = _valid
        self._wr_en     = _wr_en
        self._rd_en     = _rd_en
        self._o_empty   = _o_empty
        self._o_full    = _o_full
```

- Interface Object
 - Mimics SV-IF functionality.
 - An object contains all the signals of specific protocol
 - Makes it much easier to pass around
 - Connect method
 - Connects the given signals
- Alternative
 - CocoTB bus
 - Somewhat different paradigm

PyUVM with Interface Object -Example

```
class sat_filter_tb_base_test(uvm_test):
    def __init__(self, name="sat_filter_base_test", parent=None):
        # Declare DUT handler
        self.dut = None

        # Agent's interfaces
        self.ssdt_prod_if = None

    def build_phase(self):
        super().build_phase()

        # Create configuration objects
        self.cfg = sat_filter_tb_config.create('sat_filter_base_cfg')

        # Access the DUT through the cocotb.top handle
        self.dut = cocotb.top

        ...

        # Instantiate interface
        self.ssdt_prod_if = ssdt_interface_wrapper("ssdt_prod_if")

        # Set interfaces in configs for each component
        self.cfg.ssdt_prod_cfg.vif = self.ssdt_prod_if
```

```
def connect_phase(self):
    super().connect_phase()

    self.ssdt_prod_if.connect(
        clk_signal    = self.dut.clk,
        reset_signal  = self.dut.rst,
        valid_signal  = self.dut.in_valid,
        data_signal   = self.dut.in_data
    )
```

```
class ssdt_interface_wrapper():
    def __init__(self, clk=None,
                  rst=None,
                  name="ssdt_interface"):

        self.name    = name
        self.clk     = clk
        self.rst     = rst
        self.valid   = None
        self.data    = None

    def connect(self, clk_signal, reset_signal,
                valid_signal, data_signal):
        self.clk = clk_signal
        self.rst = reset_signal

        self.valid = valid_signal
        self.data  = data_signal
```


Running a (PyUVM) test

- Annotate test class with: `@pyuvm.test()`
- Derive from: `uvm_test`
- Test
 - `build_phase`
 - Builds testbench components
 - `connect_phase`
 - Connects testbench components
 - `run_phase`
 - Implement test
 - Usually
 - Create a virtual sequence
 - Randomize virtual sequence
 - Start virtual sequence
 - More on this later!

```
@pyuvm.test()
class AluTest(uvm_test):
    def build_phase(self):
        super().build_phase()
        self.env = AluEnv("env", self)

    def connect_phase(self):
        ...

    async def run_phase(self):
        self.raise_objection()
        super().run_phase()

        # Create, Randomize and start
        # top virtual sequence
        self.vseq = AluVSeq.create("AluVSeq")

        self.virt_sequence.randomize()

        await (self.vseq.start(self.env.vseqr))

        self.drop_objection()
```

Phase Objection

- Phase Objection is a mechanism to control the execution flow of simulation phases which takes **time – currently, only run_phase**
 - It allows components to synchronize the start and end of the run_phase
- `raise_objection()`
 - Signals to the UVM scheduler that: **I am not done!**
 - Prevents UVM scheduler to go to next phase
 - Otherwise UVM scheduler has NO way of knowing that a uvm_component is still in the run_phase
- `drop_objection()`
 - Signals to the UVM scheduler that: **I am done!**

```
from pyuvvm import *  
  
class MyComponent(uvm_component):  
    def run_phase(self, phase):  
        phase.raise_objection(self)  
        # Perform some actions  
        ...  
        phase.drop_objection(self)
```