2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Functional Coverage Closure with Python

Seokho Lee, Youngsik Kim, Suhyung Kim,
Jeong Ki Lee, Wooyoung Choe, Minho Kim

FURIOSA

accellera SYSTEMS INITIATIVE

Table of Contents

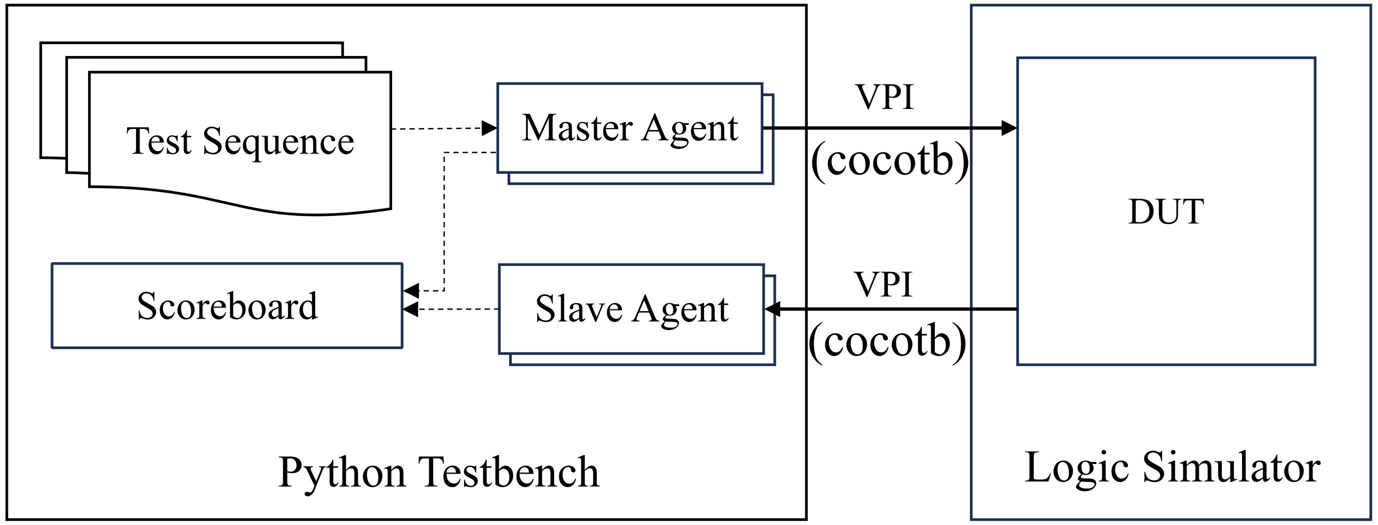
- Backgrounds
  - Python Testbench and Python extension libraries
- Motivation
  - Problems of Coverage Closure in Python Testbench
- Proposed Solution
  - Overall flow & how to guides
- Conclusion

Design Verification using Python

- Python extension libraries for verification
  - Cocotb: coroutine based cosimulation testbench environment using Python
  - PyVSC: random verification-stimulus generation
- Testbenches can be written and run by Python thanks to open-source Python libraries

Design Verification using Python

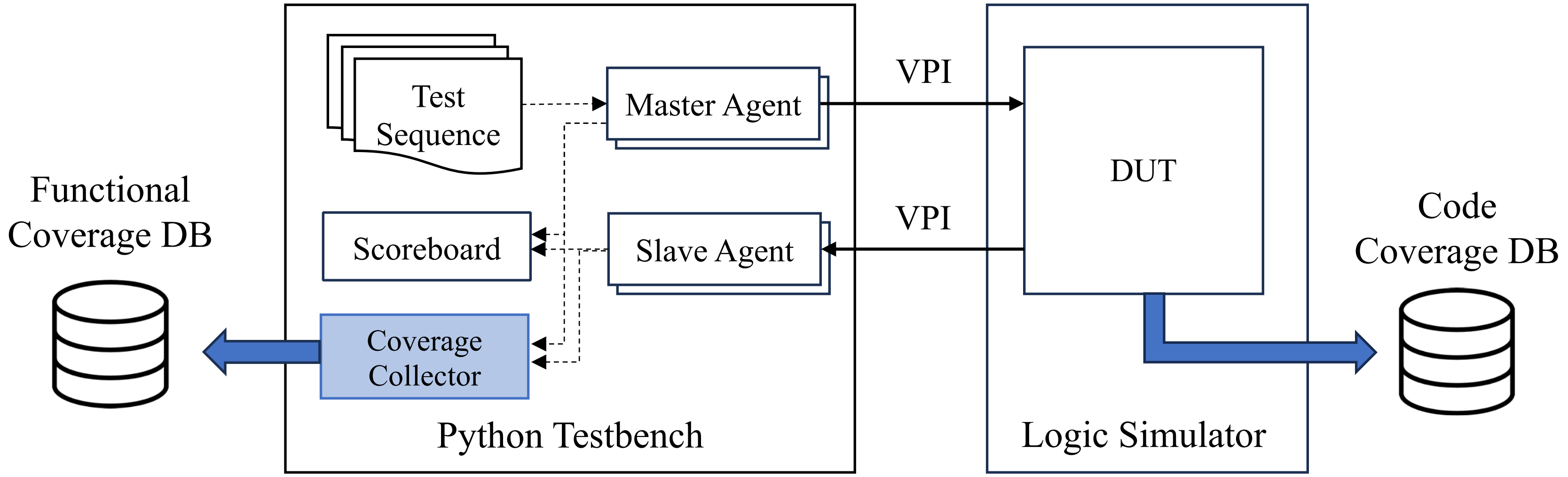
- In FuriosaAI, Most block-level testbenches are written in Python



Problems of Coverage Closure in Python (1/2)

Case 1) Measure coverage by existing Python coverage library

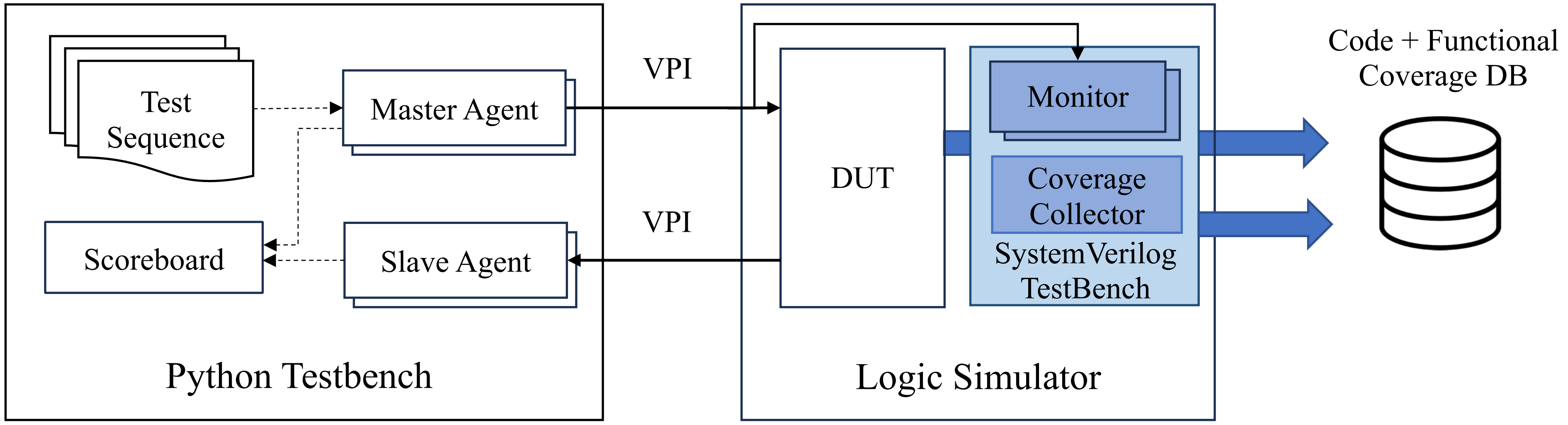
- Separate func coverage DB not compliant with legacy coverage analysis tools
- Python coverage libraries don't fully support SystemVerilog coverage features



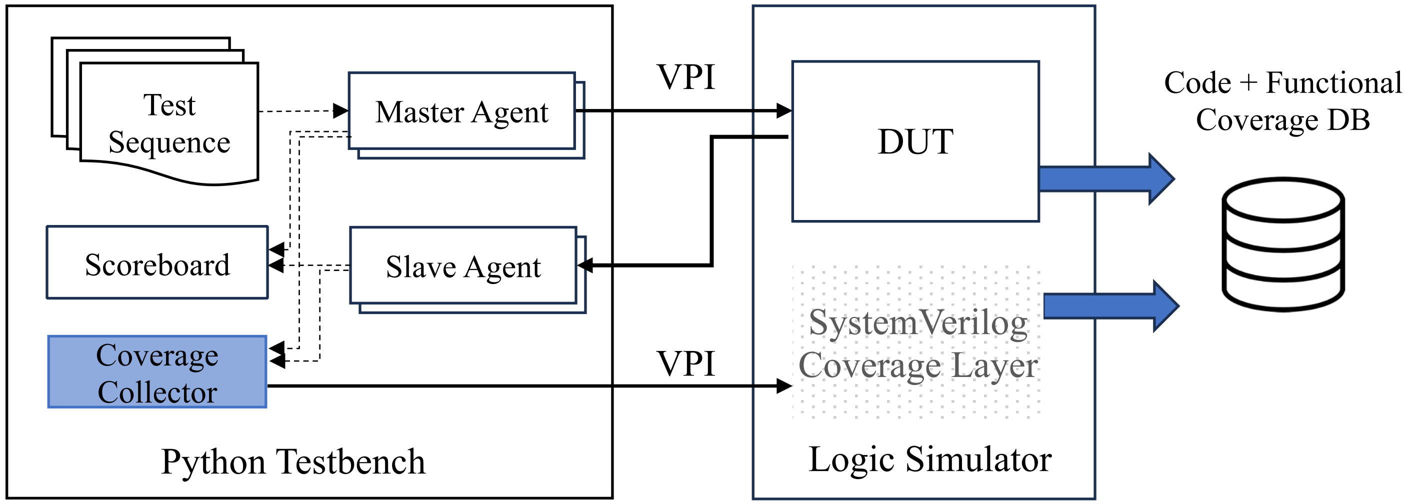
Problems of Coverage Closure in Python (2/2)

Case 2) Measure coverage by SystemVerilog + Logic Simulator

- Seperate SystemVerilog testbench with monitors and coverage collectors
- Both Python and SystemVerilog/UVM skills required

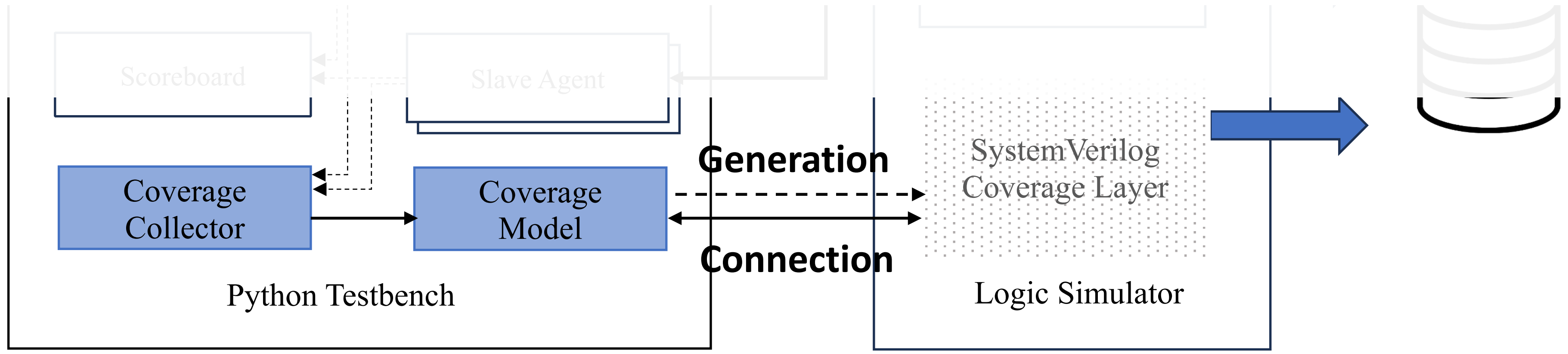


Our Proposed Solution



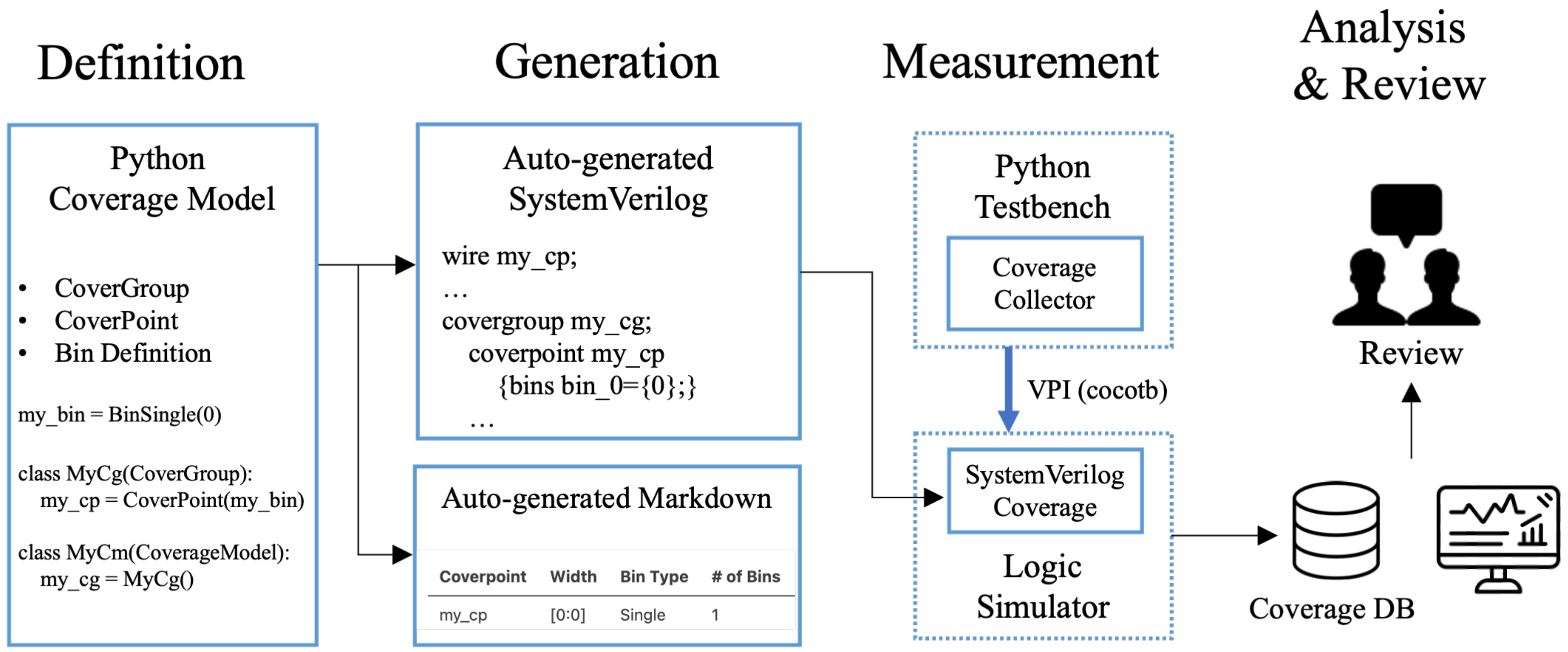
- Python testbench has coverage models and coverage collectors
- SystemVerilog coverage layer generates functional coverage database
- At sample, collected coverage data is transferred to SystemVerilog coverage layer

Our Proposed Solution



- Defining a coverage model in python allows the library to connect python with SystemVerilog
- Define coverage by SystemVerilog seamlessly

Functional Coverage Flow



Contributions

- Easy coverage definition
  - Predefined bin type for frequently used bins
  - User-defined iterator for complicated bins
  - **Coverpoint** can be defined using OOP

- Python-driven functional coverage collection
  - SystemVerilog coverage layer is generated from Python coverage model and fully controlled by Python testbench
  - SystemVerilog knowledge not required

Define Coverage Model: Predefined Bin Types

Predefined Coverage Type	Python Definition	Generated SystemVerilog Coverage

Range	cp_range_0 =
	 CoverPoint(BinRange(64))
		cp_range: coverpoint signal {
		 bins bin_0_63[64] = {[0:63]};
		}

Uniform	cp_uniform =
	 CoverPoint(BinUniform(
	 0, 128, num=5))
		cp_uniform: coverpoint signal {
		 bins bin_0_127[5] = {[0:127]};
		}

Enum	class MyEnum(IntEnum):
	 STATE_0 = 0b00
	 STATE_1 = 0b01
	 STATE_2 = 0b10
	 STATE_3 = 0b11
	
	cp_enum =
	 CoverPoint(BinEnum(MyEnum))
		cp_enum: coverpoint signal {
		 bins STATE_0 = {0};
		 bins STATE_1 = {1};
		 bins STATE_2 = {2};
		 bins STATE_3 = {3};
		}

Predefined Coverage Type	Python Definition	Generated SystemVerilog Coverage

Exponential	cp_exp =
	 CoverPoint(BinExp(16))
		cp_exp: coverpoint signal {
		 bins bin_0 = {0};
		 bins bin_1_1 = {[1:1]};
		 bins bin_2_3 = {[2:3]};
		 bins bin_4_7 = {[4:7]};
		 bins bin_8_15 = {[8:15]};
		}

Bitwise	cp_bitwise =
	 CoverPoint(BinBitwise(4))
		cp_bitwise_0: coverpoint signal[0];
		cp_bitwise_1: coverpoint signal[1];
		cp_bitwise_2: coverpoint signal[2];
		cp_bitwise_3: coverpoint signal[3];

Transition	cp_transition =
	 CoverPoint(BinTransition(
	 (2,5), (2,10), (3,8)
	))
		cp_onehot: coverpoint signal {
		 bins bin_2_5 = (2=>5);
		 bins bin_2_10 = (2=>10);
		 bins bin_3_8 = (3=>8);
		}

Define Coverage Model: User-defined Iterator Types

- Complicated coverage bins can be easily expressed

```python
def narrow_strobe(bus_byte_width):
    for i in range(log2Ceil(bus_byte_width)):
        size = 1 << i
        for j in range(bus_byte_width):
            strobe = ((1 << size) - 1) << (j * size)
            yield (f"en{size}byte_{hex(strobe)}", strobe)


cp_strobe = CoverPoint(narrow_strobe(8))
```

```systemverilog
cp_strobe: coverpoint cg_axi_narrow_cp_strobe {
  bins en1byte_0x1 = {'b1};
  bins en1byte_0x2 = {'b10};
  bins en1byte_0x4 = {'b100};
  bins en1byte_0x8 = {'b1000};
  bins en1byte_0x10 = {'b10000};
  bins en1byte_0x20 = {'b100000};
  bins en1byte_0x40 = {'b1000000};
  bins en1byte_0x80 = {'b10000000};
  bins en2byte_0x3 = {'b11};
  bins en2byte_0xc = {'b1100};
  bins en2byte_0x30 = {'b110000};
  bins en2byte_0xc0 = {'b11000000};
  bins en4byte_0xf = {'b1111};
  bins en4byte_0xf0 = {'b11110000};
}
```

Implementing Coverage Collector

- connect
  Connect Python coverage instances to SystemVerilog layer during initialization
- assign (<=)
  Assign values to coverpoints
- sample
  Trigger a sampling event

```
class MyCoverageCollector:
  def __init__(self, dut):
    self.my_cov = MyCoverageModel()
    cov_inst = getattr(dut, self.my_cov.sv_instname)

    # connect python coverage objects
    self.my_cov.connect(cov_inst)

  def collect_coverage(self, coverpoint_value):
    # assign sampling values
    self.my_cov.my_cg.my_cp <= coverpoint_value

    # trigger sampling
    self.my_cov.my_cg.sample()
```

How to implement coverage collector

- connect
  connect coverage instance to System Verilog when simulation launched.

```
module my_cov ();
  wire [1:0] my_cg_my_cp;
  wire my_cg_sample;
  covergroup my_cg;
    my_cp: coverpoint my_cg_my_cp {
      bins bin_0 = {0}; bins bin_1 = {1};
      bins bin_2 = {2}; bins bin_3 = {3};
    }
  endgroup : my_cg
  my_cg my_cg_inst = new;
  always @(my_cg_sample) begin
    my_cg_inst.sample();
  end
endmodule
```

```
class MyCoverageCollector:
 def __init__(self, dut):
   cov_inst = getattr(dut, self.my_cov.sv_instname)
   self.my_cov = MyCoverageModel()

   # connect python coverage objects
   self.my_cov.connect(cov_inst)
```

```
class MyCoverageModel(CoverageModel):
   class MyCoverGroup(CoverGroup):
       my_cp = CoverPoint([0, 1, 2, 3])

   my_cg = MyCoverGroup()

my_cov = MyCoverageModel()
```

How to implement coverage collector

- assign (<=)
  Assign values to coverpoints

```
module my_cov ();
  wire [1:0] my_cg_my_cp;
  wire my_cg_sample;
  covergroup my_cg;
    my_cp: coverpoint my_cg_my_cp {
      bins bin_0 = {0}; bins bin_1 = {1};
      bins bin_2 = {2}; bins bin_3 = {3};
    }
  endgroup : my_cg
  my_cg my_cg_inst = new;
  always @(my_cg_sample) begin
    my_cg_inst.sample();
  end
endmodule
```

```
class MyCoverageCollector:
 def __init__(self, dut):
   cov_inst = getattr(dut, self.my_cov.sv_instname)
   self.my_cov = MyCoverageModel()

   # connect python coverage objects
   self.my_cov.connect(cov_inst)

 def collect_coverage(self, coverpoint_value):
   # assign sampling values
   self.my_cov.my_cg.my_cp <= coverpoint_value
```

How to implement coverage collector

- sample
  sampling by flip sampling trigger signal

```
module my_cov ();
  wire [1:0] my_cg_my_cp;
  wire my_cg_sample;
  covergroup my_cg;
    my_cp: coverpoint my_cg_my_cp {
      bins bin_0 = {0}; bins bin_1 = {1};
      bins bin_2 = {2}; bins bin_3 = {3};
    }
  endgroup : my_cg
  my_cg my_cg_inst = new;
  always @(my_cg_sample) begin
    my_cg_inst.sample();
  end
endmodule
```

```
class MyCoverageCollector:
 def __init__(self, dut):
   cov_inst = getattr(dut, self.my_cov.sv_instname)
   self.my_cov = MyCoverageModel()

   # connect python coverage objects
   self.my_cov.connect(cov_inst)

 def collect_coverage(self, coverpoint_value):
   # assign sampling values
   self.my_cov.my_cg.my_cp <= coverpoint_value

   # trigger sampling
   self.my_cov.my_cg.sample()
```

Results

- Functional coverage closure of FuriosaAI's latest AI chip
  - Productivity increase by 4 ~ 20 times in terms of line of codes

	# of Covergroups	# of Bins	Python code lines	SystemVerilog code lines
Block A	152	7294	395	6939
Block B	22	10854	310	1238
Block C	12	14900	64	1091

Conclusion

- Propose functional coverage library in Python testbench.
- The library has the programmability of python
  and the rich functional coverage features of SystemVerilog
- We successfully close functional coverage using the library

Resources

- Cocotb extension for functional coverage closure
  - https://github.com/furiosa-ai/cocotbext-fcov
- Examples
  - https://github.com/furiosa-ai/dvcon2024-functional-coverage-closure-with-python

Questions?