AEDVICES Consulting
166B Rue du Rocher de Lorzier
38430 Moirans– France
trainings@aedvices.com
http://www.aedvices.com

Training on
IP & SoC Functional Verification Methodology
Using UVM

LAB on
Functional Coverage

Objectives

This lab goes through the main concepts of functional coverage. It introduces different types of functional covergroup and coverpoints as a verification target.

Introduction

This lab proposes to analyse the coverage results of randomly generated transaction seen in previous lab.

Instructions

Follow instructions given in “aedv_training_labs_intructions_for_questa.pdf”.
Open the file <SANDBOX>/labs-Xdays/labNN-systemverilog_functional_coverage /lab.sv
Select

Global Explanation

Until now, we have generated transactions with random attributes (address, data, etc), but we did not know which values we achieved nor how many times we achieved them, now the student will learn how to use functional coverage in SystemVerilog to be sure about what is being covered.

To do this, we will modify the *adderTransactionBase* class to add a convergroup and than manipulate the coverpoints to get the coverage desired.

Step 1 – Analyse coverage

Open the file
<SANDBOX>/labs-Xdays/labNN-systemverilog_functional_coverage /project_utils_pkg.sv

Search for “LAB-TODO-STEP1”

A first covergroup is defined as followed:

```
covergroup trans_cg;

    in_data_cover   : coverpoint in_data;
    addr_cover      : coverpoint addr;
    dir_cover       : coverpoint dir;
    action_cover    : coverpoint action;

endgroup
```

In this example, the covergroup is sampled in the post_randomize() method so that we collect coverage information from all generated transactions:

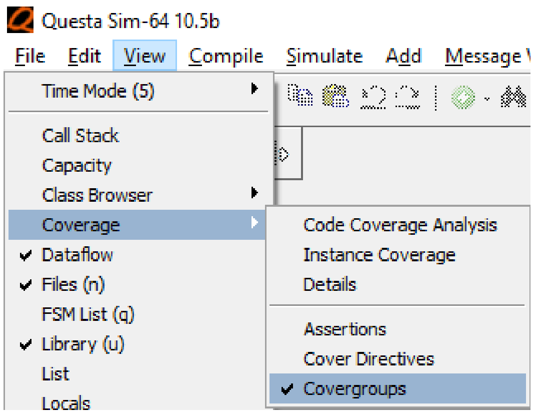
```
// LAB-TODO-STEP-1
function new ();
    trans_cg = new();
endfunction

// LAB-TODO-STEP-1
function void post_randomize ();
    trans_cg.sample();
endfunction
```

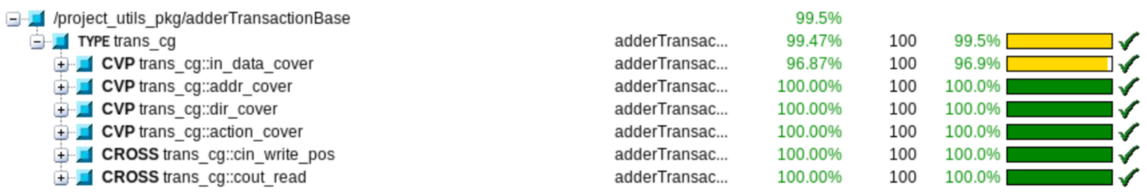
Recompile and Reload the test.

Open the Coverage Analysis Window:

Using Mentor Questa:

- Select the menu View Coverage Covergroups

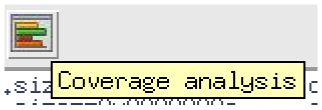




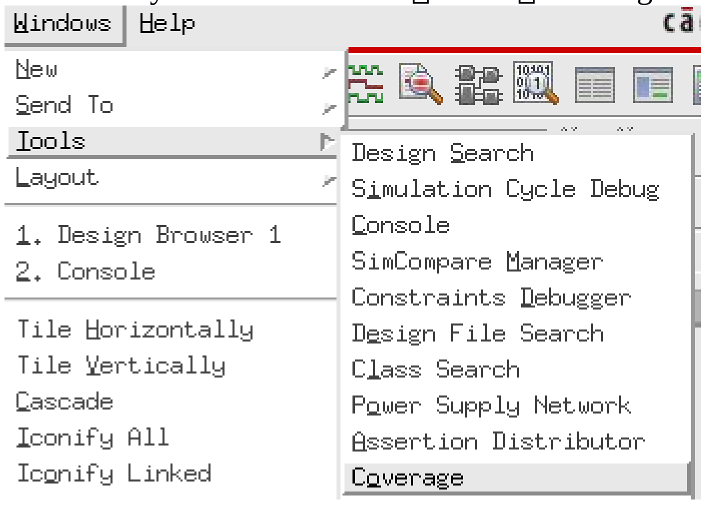
/project_utils_pkg/adderTransactionBase		99.5%			
TYPE trans_cg	adderTransac...	99.47%	100	99.5%	✓
CVP trans_cg::in_data_cover	adderTransac...	96.87%	100	96.9%	✓
CVP trans_cg::addr_cover	adderTransac...	100.00%	100	100.0%	✓
CVP trans_cg::dir_cover	adderTransac...	100.00%	100	100.0%	✓
CVP trans_cg::action_cover	adderTransac...	100.00%	100	100.0%	✓
CROSS trans_cg::cin_write_pos	adderTransac...	100.00%	100	100.0%	✓
CROSS trans_cg::cout_read	adderTransac...	100.00%	100	100.0%	✓

Using Cadence Incisive:

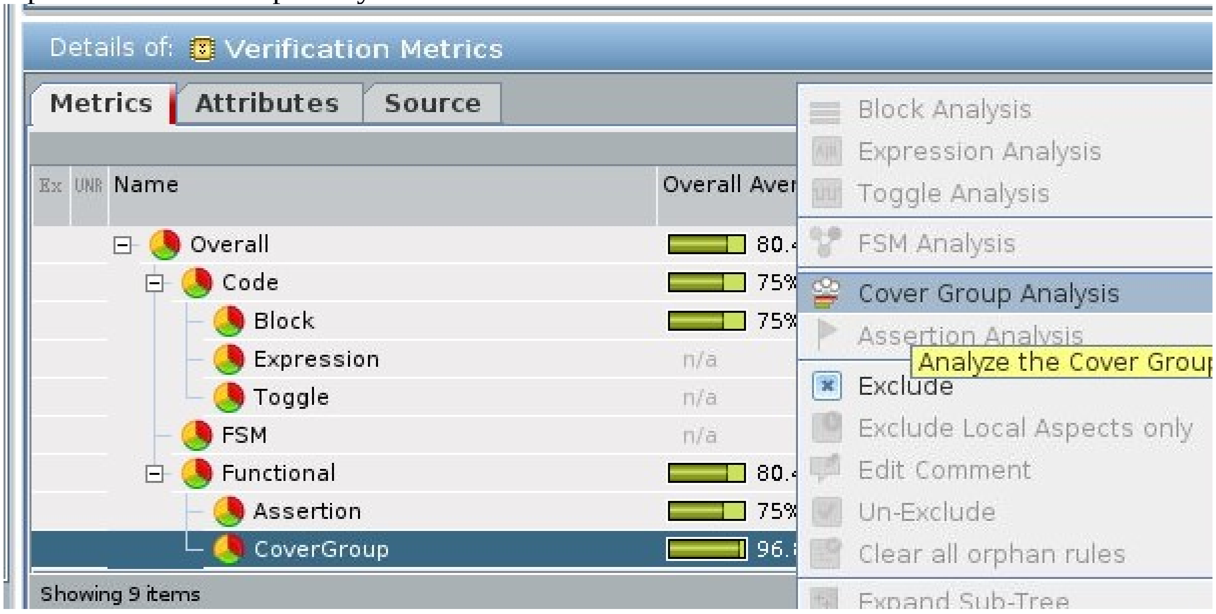
- Click on the “coverage analysis” icon

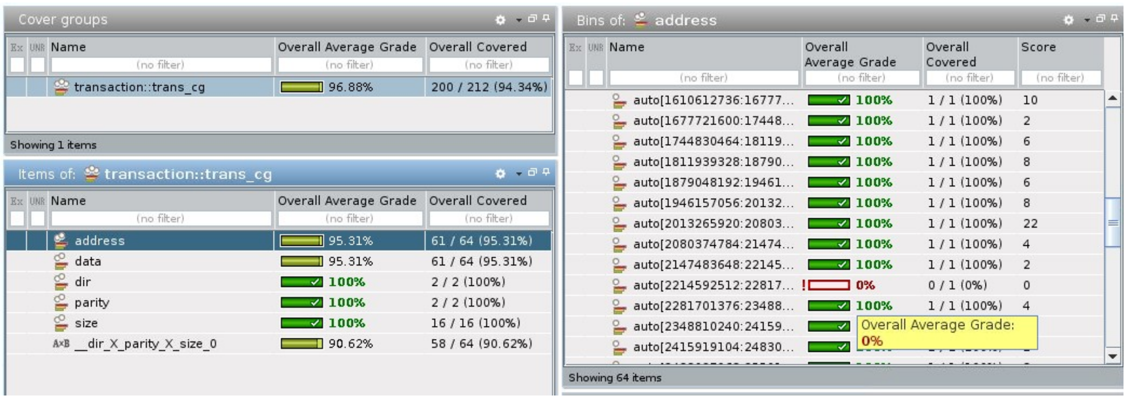


- Alternatively follow: Windows □ Tools □ Coverage



- Open the CoverGroup Analysis





Cover groups

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		transaction::trans_cg	96.88%	200 / 212 (94.34%)

Showing 1 items

Items of: transaction::trans_cg

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		address	95.31%	61 / 64 (95.31%)
		data	95.31%	61 / 64 (95.31%)
		dir	✓ 100%	2 / 2 (100%)
		parity	✓ 100%	2 / 2 (100%)
		size	✓ 100%	16 / 16 (100%)
		A×B __dir_X_parity_X_size_0	90.62%	58 / 64 (90.62%)

Bins of: address

Ex	UNR	Name	Overall Average Grade	Overall Covered	Score
		(no filter)	(no filter)	(no filter)	(no filter)
		auto[1610612736:16777…	✓ 100%	1 / 1 (100%)	10
		auto[1677721600:17448…	✓ 100%	1 / 1 (100%)	2
		auto[1744830464:18119…	✓ 100%	1 / 1 (100%)	6
		auto[1811939328:18790…	✓ 100%	1 / 1 (100%)	8
		auto[1879048192:19461…	✓ 100%	1 / 1 (100%)	6
		auto[1946157056:20132…	✓ 100%	1 / 1 (100%)	8
		auto[2013265920:20803…	✓ 100%	1 / 1 (100%)	22
		auto[2080374784:21474…	✓ 100%	1 / 1 (100%)	4
		auto[2147483648:22145…	✓ 100%	1 / 1 (100%)	2
		auto[2214592512:22817…	! 0%	0 / 1 (0%)	0
		auto[2281701376:23488…	✓ 100%	1 / 1 (100%)	4
		auto[2348810240:24159…	✓ [illegible]	[illegible]	[illegible]
		auto[2415919104:24830…	✓ [illegible]	[illegible]	[illegible]

Overall Average Grade:
0%

Showing 64 items

Step 2 – Specifying input data coverage

Search for "LAB-TODO-STEP-2"
Note that the data input is sampled always even if there is no writing transaction. So, we will use the *iff* keyword to specify when the input data should be sampled.

```
in_data_cover  : coverpoint in_data iff (dir == WRITE && action == ACC);
```

Re-Compile
Re-Load & Rerun the simulation

Step 3 – Limit coverage to the register addresses

Our DUT has just 6 registers, so the valid addresses are in the range: {0:5}

Search for "LAB-TODO-STEP3"

- Define Coverage Bins as followed:

```
addr_cover     :    coverpoint addr{
    bins valid_addr[] = {[0:5]};
}
```

Re-Compile
Re-Load & Rerun the simulation

Step 4 – Carry In set coverage

Search for "LAB-TODO-STEP-4"

We want to cover when the *cin* register is set, so we must cover the writing of a value of *1* in the *0x03* address. To do this, we can add the following cross cover point:

```
cin_write_pos  : cross in_data_cover, addr_cover{
    ignore_bins ignore_addr    = ! binsof(addr_cover.valid_addr) intersect{3};
    ignore_bins ignore_in_data = ! binsof(in_data_cover)         intersect {1};
}
```

Q: Why do not we need to do a cross with the *dir*(direction) and action variable?

Step 4 – Carry Out reading coverage

Now you are invited to perform a coverage that covers the *cout* register reading. Use the Carry In cover point as example.

Step 6 – Improve test generation

Q: What is not covered ?
Q: How can you improve this coverage ?

Search for "LAB-TODO-STEP-6" in the file <SANDBOX>/labs-Xdays/labNN-systemverilog_functional_coverage /lab_prog.sv

Set the for loop to execute 100 times.

Re-Compile
Re-Load & Rerun the simulation