

MCDF 实验 5

嗨！首先要恭喜你，终于坚持到我们最后这一个实验了！

我们本次实验将带领大家认识如何定义覆盖率，如何从验证计划到测试用例的实现，最后再到覆盖率的量化。从本次实验，大家可以掌握验证量化的两种基本数据，即代码覆盖率和功能覆盖率。从这两种覆盖率,我们就可以掌握何时结束验证，确认验证的完备性。那么，接下来就让我们开始吧。

首先请下载实验 5 的代码。实验 5 的代码是基于实验 4 的代码，主要修改的文件即 `mcd_f_pkg.sv` 在 `mcd_f_pkg.sv` 中，路桑已经为大家添加了一个关于 MCDF 的覆盖率模型 `mcd_f_coverage`,并且将它例化在顶层环境中。你可以阅读代码，了解 `mcd_f_coverage` 的例化、虚接口的传递、覆盖率的定义和采样。

同时，路桑为大家提供了一个新的 test,即 `mcd_f_full_random_test`. 这个 test 尽可能的将一些测试相关参数在仿真时进行了随机化。在接下来的仿真中，你依然可以复用你之前实验 4 按照要求创建的几个 test,将它们搬迁到 lab5 中。不过,我们实验 5 的要求是，需要同学们最终达到“尽可能高的代码覆盖率和功能覆盖率”。

通过等级:代码覆盖率大于 90%，功能覆盖率大于 90%。

优秀等级:代码覆盖率大于 95%，功能覆盖率等于 100%。

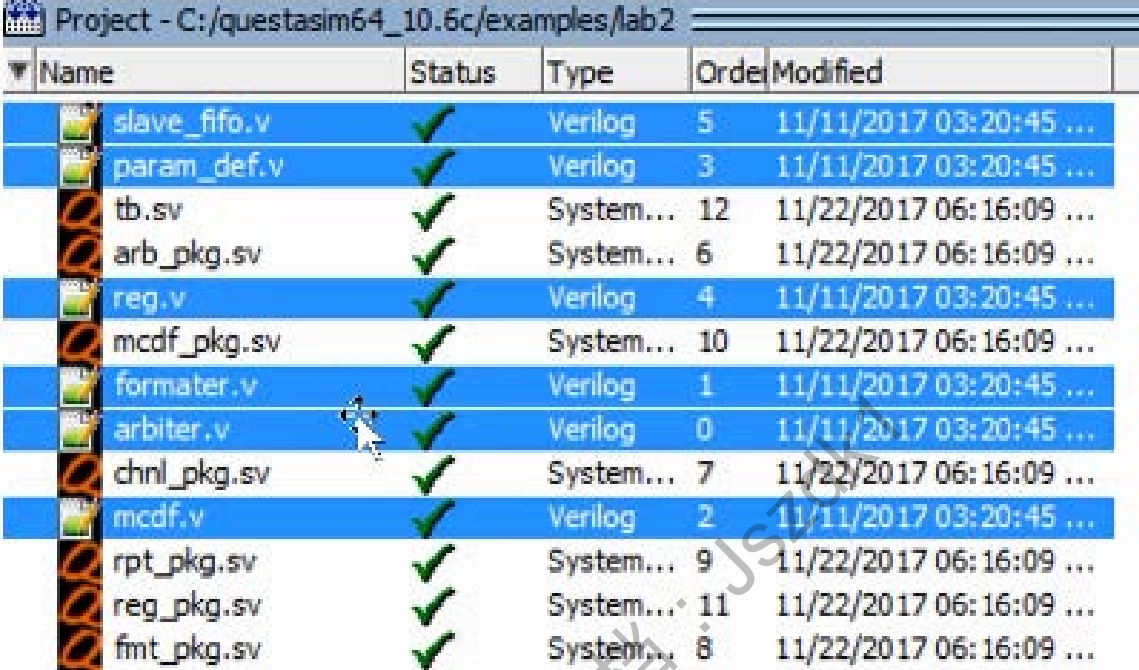
所以结合我们本次实验最终的验收目标，你可以复用你之前的测试用例，也可以使用路桑的测试用例。当然，在你学习了覆盖率相关的课程之后，你就懂得了，当最终覆盖率无法提升时，需要修改你的约束，或者创建新的 test。

编译

在编译过程中，我们需要对于设计相关的文件设置额外的覆盖率编译选项。如下面的

截图:

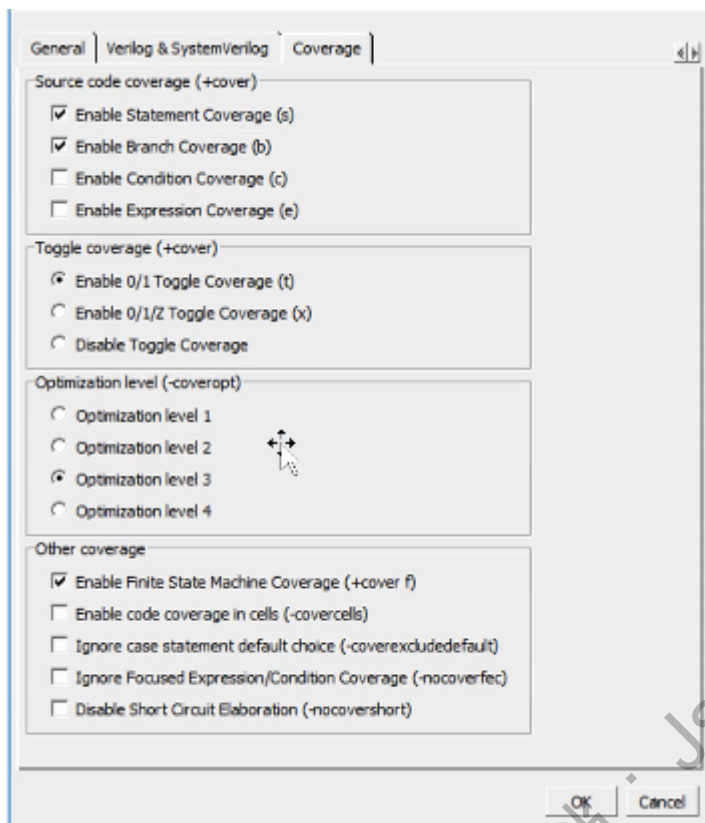
1、只选择与设计有关的文件



Name	Status	Type	Order	Modified
slave_fifo.v	✓	Verilog	5	11/11/2017 03:20:45 ...
param_def.v	✓	Verilog	3	11/11/2017 03:20:45 ...
tb.sv	✓	System...	12	11/22/2017 06:16:09 ...
arb_pkg.sv	✓	System...	6	11/22/2017 06:16:09 ...
reg.v	✓	Verilog	4	11/11/2017 03:20:45 ...
mcd_f_pkg.sv	✓	System...	10	11/22/2017 06:16:09 ...
formater.v	✓	Verilog	1	11/11/2017 03:20:45 ...
arbiter.v	✓	Verilog	0	11/11/2017 03:20:45 ...
chnl_pkg.sv	✓	System...	7	11/22/2017 06:16:09 ...
mcd_f.v	✓	Verilog	2	11/11/2017 03:20:45 ...
rpt_pkg.sv	✓	System...	9	11/22/2017 06:16:09 ...
reg_pkg.sv	✓	System...	11	11/22/2017 06:16:09 ...
fnt_pkg.sv	✓	System...	8	11/22/2017 06:16:09 ...

2. 点击右键，选择 compile -> compile properties, 在弹出设置栏的 coverage 一栏中，

如图选择以下选项，然后点击 OK。



3.完成所有文件的编译"Compile All"。这一步将在编译 DUT 文件时生成代码覆盖率的模型，而我们之所以没有给 TB 相关文件添加代码覆盖率选项，是由于测试平台的覆盖率不是我们需要关注的对象。

注意:与测试相关的文件不要设置覆盖率编译选项。

仿真

接下来，在仿真窗口(transcript) 中，可以参考路桑的仿真命令:

```
vsim -i -classdebug -solvefaildebug -coverage -coverstore
```

```
C:/questasim64 10.6c/examples -testname mcdf_full_random_test -sv_seed
```

```
random +TESTNAME=mcdf_full_random_test -l mcdf_full_random_test.log
```

```
work.tb.
```

注：此处-l 中的 l 为小写的 L，不是符号“|”。

这里需要注意的是标注黄色的仿真命令,这些新增的命令说明如下：

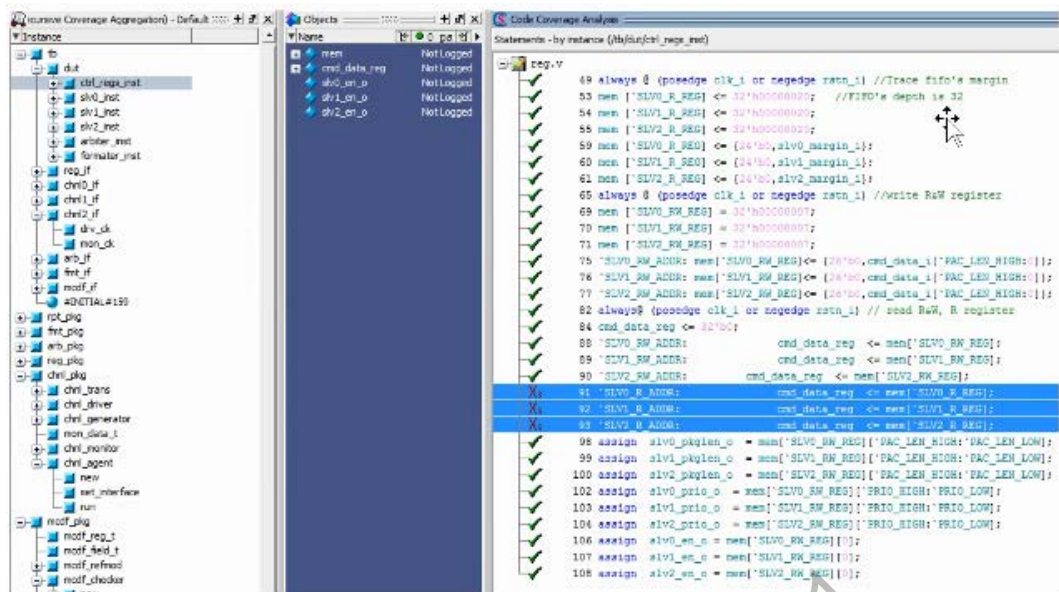
- coverage:会在仿真时产生代码覆盖率数据，功能覆盖率数据则默认会生成，与此选项无关。

- coverstore COVERAGE_STORAGE_PATH:这个命令是用来在仿真在最后结束时，生成覆盖率数据并且存储到 COVERAGE_STORAGE_PATH。你可以自己制定 COVERAGE_STORAGE_PATH，但需要注意路径名中不要包含中文字符。

- testname. TESTNAME:这个选项是你需要添加本次仿真的 test 名称，你可以使用同+TESTNAME 选项一样的 test 名称。这样在仿真结束后，将在 COVERAGE_STORAGE_PATH 下产生一个覆盖率数据文件 "{TESTNAME}_{SV_SEED}.data"。由于仿真时我们传入的种子是随机值，因此我们每次提交测试，在测试结束后都将产生一个独一无二的覆盖率数据。例如 mcdf_full_random_test_1988811153.data。

接下来在仿真窗口敲入命令“run -all”，在仿真最后自动结束时(大约 10ms)会弹出仿真要求结束的对话框“Are you sure you want to finish?”，你可以点击 NO，但一定要选择菜单栏“Simulate -> end simulation”来结束本次仿真。只有你结束了本次仿真,你才会得到上面提到的覆盖率数据，例如 mcdf full random_test 1988811153.data。

在本次仿真过程中或者结束时，你也可以利用仿真器直接查看代码覆盖率或者功能覆盖率。首先你需要选中 View -> Coverage -> "Code Coverage Analysys"和"Covergroups"。如果你需要查看代码覆盖率，那么选择新添加的 Analysis 窗口，然后逐个点击 Sim 窗口中 DUT 层次中的个别模块，例如下图中点击“ctrl regs. Inst”你可以在 Analysis 窗口中看到哪些代码被执行了而哪些代码在本次仿真中没有执行过。



如果你需要查看功能覆盖率，那么你可以在新添加的“Covergroups”窗口中查看本次仿真所收集到的功能覆盖率。

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
cg_mcdf_reg_write_read	mcdcf_cover...	66.6%	100	66.6%	✓	✓
CVP cg_mcdf_reg_write_read::addr	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_mcdf_reg_write_read::cmd	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CROSS cg_mcdf_reg_write_read::cmdXaddr	mcdcf_cover...	66.6%	100	66.6%	✓	✓
TYPE cg_mcdf_reg_legal_access	mcdcf_cover...	41.6%	100	41.6%	✓	✓
CVP cg_mcdf_reg_legal_access::addr	mcdcf_cover...	33.3%	100	33.3%	✓	✓
CVP cg_mcdf_reg_legal_access::cmd	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CVP cg_mcdf_reg_legal_access::data	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_mcdf_reg_legal_access::data	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CROSS cg_mcdf_reg_legal_access::cmdXaddrXdata	mcdcf_cover...	41.6%	100	41.6%	✓	✓
TYPE cg_channel_disable	mcdcf_cover...	61.1%	100	61.1%	✓	✓
CVP cg_channel_disable::ch0_en	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_channel_disable::ch1_en	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_channel_disable::ch2_en	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_channel_disable::ch0_vld	mcdcf_cover...	50.0%	100	50.0%	✓	✓
CVP cg_channel_disable::ch1_vld	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CVP cg_channel_disable::ch2_vld	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CROSS cg_channel_disable::ch0ch1ch2	mcdcf_cover...	61.1%	100	61.1%	✓	✓
TYPE cg_arbiter_priority	mcdcf_cover...	25.0%	100	25.0%	✓	✓
CVP cg_arbiter_priority::ch0_prio	mcdcf_cover...	25.0%	100	25.0%	✓	✓
CVP cg_arbiter_priority::ch1_prio	mcdcf_cover...	25.0%	100	25.0%	✓	✓
CVP cg_arbiter_priority::ch2_prio	mcdcf_cover...	25.0%	100	25.0%	✓	✓
TYPE cg_formatter_length	mcdcf_cover...	29.1%	100	29.1%	✓	✓
CVP cg_formatter_length::vid	mcdcf_cover...	33.3%	100	33.3%	✓	✓
CVP cg_formatter_length::length	mcdcf_cover...	25.0%	100	25.0%	✓	✓
TYPE cg_formatter_grant	mcdcf_cover...	100.0%	100	100.0%	✓	✓
CVP cg_formatter_grant::delay_req_to_grant	mcdcf_cover...	100.0%	100	100.0%	✓	✓

合并覆盖率

你可以参考上面的仿真步骤，运行不同的仿真，或者运行同一个 test，它们都会生成独一无二的数据库。接下来，你就可以将之前统一在 COVERAGE_STORAGE_PATH 下面生成的 xxx.data 覆盖率数据做合并了。你可以在 Questasim 的仿真窗口中敲入命令。

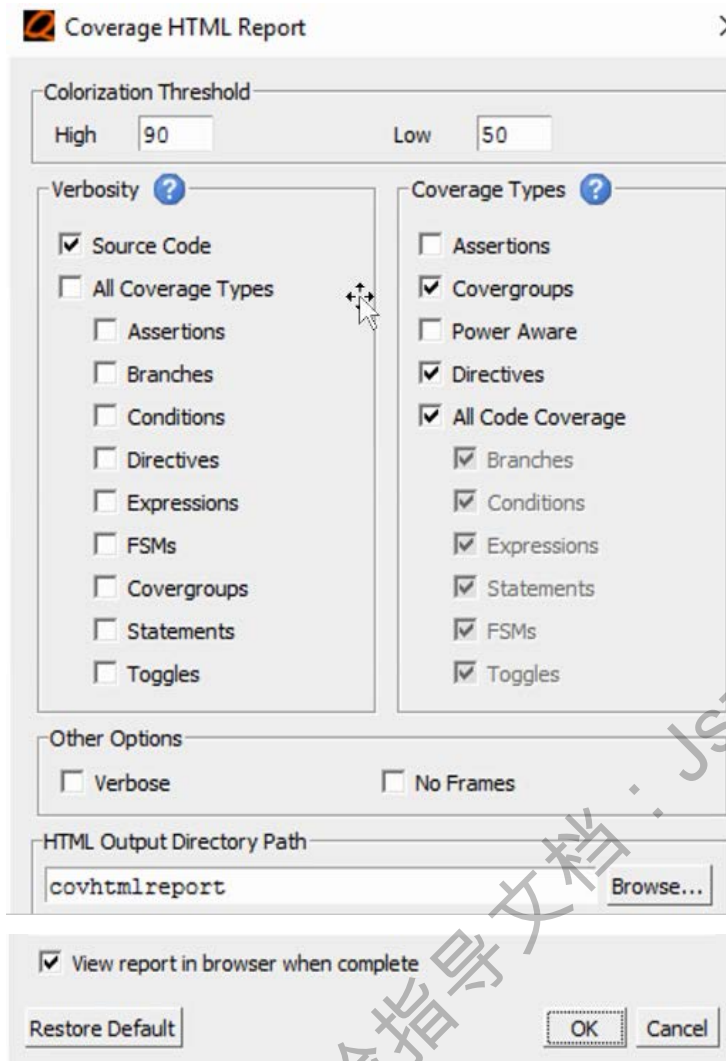
●vcover merge -out merged_coverage.ucdb C:/questasim64 10.6c/examples

这里标注黄色的部分依然代表 COVERAGE_STORAGE_PATH。这个命令即是将你之前产生的若干个 xxx.data 的覆盖率合并在一起，生成一个合并在一起的覆盖率文件。所以，在测试前期，你提交的测试越多，那么理论上覆盖率的增长也就越明显。

接下来，你可以点击 File -> Open 来打开这个合并后的 UCDB 覆盖率数据库(注意选择文件类型 UCDB 就可以看到这个文件了)。当你打开这个数据库之后，你可以发现合并后的数据库要比之前单独提交的任何一个测试在仿真结束时的该次覆盖率都要高。例如你可以在 covergroups 窗口栏中查看功能覆盖率,也可以在 Analysis 窗口中查看代码覆盖率。

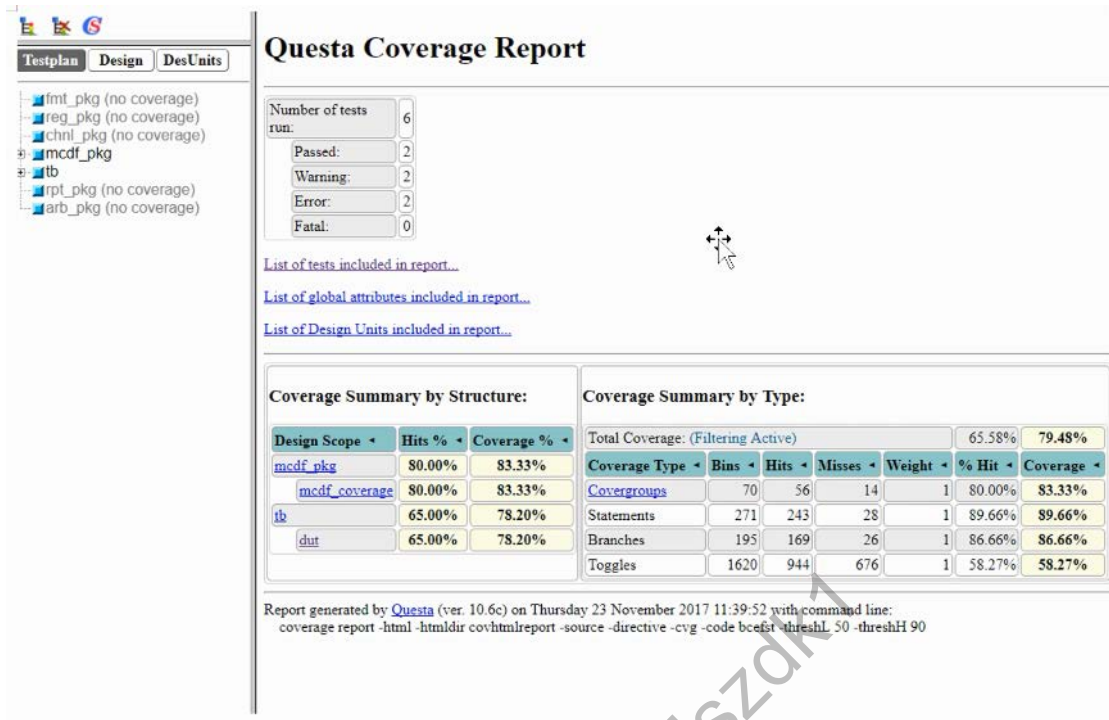
分析覆盖率

你可以依旧使用 Questasim 来打开 UCDB 利用工具来查看覆盖率，或者更直观的方式是在打开当前覆盖率数据库的同时，生成 HTML 报告。选择 Tools -> Coverage Report-> HTML，按照下图所示进行勾选：



单击 OK 后，Questasim 就会帮助你生成一份详尽的 HTML 覆盖率文档。由于我们勾选了这些内容：

- Covergroups
- Statements。
- Branches。
- Toggles。



所以接下来你需要让 DUT 的代码覆盖率和 mcd_f_coverage 的功能覆盖率达到我们最终的验收要求。你可以选用路桑的 mcd_f_full_random_test 或者复用你之前在实验 4 中实现的测试。当你发现测试无法再提高覆盖率时，你需要修改约束或者创建新的测试最终来达到我们验证完备性的要求。

祝你好运！