# 赛题解析

（赛题：基于 VCD 的 FSM 覆盖率统计）

## 一 赛题背景

验证覆盖率是衡量数字验证质量的重要指标，FSM 覆盖作为其中重要一环，重点对代码验证过程中的状态和状态转移覆盖状况进行评价。通过检查各状态和状态转移是否被覆盖到，我们可以检查预期的功能是否被覆盖；相反的，通过检查是否有不在设计范围内的状态或状态转移被意外覆盖到，进而检测到设计中的漏洞。

FSM 覆盖率通常包括两部分内容，第一部分是 FSM（有限状态机）识别和模型提取，该模型描述了有限个状态以及这些状态之间的转移行为；第二部分是 FSM 覆盖情况，即提取的状态和状态转移模型是否在仿真过程中被覆盖到。值得注意的是，第一部分有限状态机识别发生在静态分析的过程中，识别的是可能的状态和可能的状态转移，并不代表被识别的状态或状态转移一定能在仿真过程中被覆盖，这一点需要跟第二步区分开来。

对于第一步状态机的识别，已使用芯华章 Galaxsim 工具的自动识别并提取 FSM，并通过 YAML 文件提供 FSM 相关信息。FSM 信息包括信号名称、状态名称、状态值和状态转移情况。

为了更好的理解 FSM 识别，图 1 和图 2 分别给出 Verilog 源文件结构示例及对应的 YAML 文件表达的 FSM。

```
1  // RTL
2  `timescale 1ns/1ns
3  module test_fsm(clk, reset, detect)
4  input clk, reset, detect;
5  reg [1:0] current, next;
6  parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
7
8  always @(clk)
9  case(current)
10   S0: begin
11      if (detect) next = S1;
12      else next = S3;
13   end
14   S1: next = S2;
15   default: next = S0;
16 endcase
17
18 always @(posedge clk)
19   if (reset) current = S0;
20   else current = next;
21
22 endmodule
```

```
1  //Testbench:
2  `timescale 1ns/1ns
3  module test;
4  reg clk, reset, detect;
5
6  always #50 clk = ~clk;
7
8  initial begin
9    clk = 0;
10   reset = 0;
11   detect = 1;
12   #400 $finish;
13 end
14
15 test_fsm test_fsm1(clk, reset, detect);
16
17 endmodule
```

**图** 1 RTL **代码与** Testbench **举例**

在图 1 中，由第 9 行到 16 行的 case 语句块可以判断，当前 current 的值决定了信号 next 的值，由 18 行到 20 行的 always 语句块可知，每次有上升沿的信号 clk，如果 reset =true，信号 current 会在 19 行被重置成 S0；否则，在 20 行，next 的值又会被赋给 current。我们由此不难发现这个 FSM 的几个要素：

1、信号 current 被用来追踪当前信号的值；

2、信号 current 可能达到的几个状态值，分别是 S0，S1，S2，S3；

3、信号 next 作为用来给信号 current 传递值的媒介，可将下个状态值传递给 current；

4、信号 current 在接收到下个状态值时，产生状态转移。

图 2 给出 YAML 文件表达的 FSM 结构。

```
1  FSMCONFIG:
2   -FSM: current
3    LINKS:
4      - next
5    MODULE: test_fsm
6    STATES:
7      - S3: 3
8      - S2: 2
9      - S0: 0
10     - S1: 1
11   TRANSITIONS:
12     - S1->S2
13     - S0->S1
14     - S3->S0
15     - S2->S0
16     - S1->S0
17     - S0->S3
```

**图** 2 Galaxsim **工具生成的** FSM YAML **文件**

由图 2 中可以看到提取的 FSM 结构。第 2 行给出有限状态机的信号名称，即 current;第 3 行描述能够直接或间接给 FSM 的 current 信号进行赋值的信号(这里是 next，可能有多个）。第 5 行 MODULE 是当前 FSM 信号所在的模块名字。第 6 行 STATES 语句表明下面的 4 行为 FSM 所有状态的名字及对应值；第 11 行 TRANSITIONS 语句说明下面的 6 行包含所有状态转移，如从状态 S1 迁移到 S2。

由图 1 中 RTL 代码的第 10，11 行可知，当 current = S0 且 detect=0 时，信号 current 将发生状态转移 S0->S1,（next 信号会被赋值，并在某个时机，即在第 20 行，信号 next 又会赋值给 current）；由第 12 行可知，当 current = S0 且 detect != 0 时，信号 current 可以发生状态转移 S0->S3；由第 14 行可知,当 current = S1 时,信号 current 可发生状态转移 S1->S2;第 15 行 default 语句可知，只有当 current 不等于 S0 和 S1 时才会进入该语句，而本例考虑的信号 current 的状态总共有 S0，S1，S2，S3 四种，所以只考虑 current=S2 或 S3 两种状态时进入 15 行的 default 语句，信号 current 在 15 行会有两个可能的状态转移：S2->S0，S3-> S0。剩下几个状态转移均由第 19 行的 reset 分支产

生，任何当前状态都有可能被重置成 S0，本赛题不考虑自身转移，即 S0->S0，所以此处产生 S1->S0，S2-S0，S3->S0 三种状态转移。整个 RTL 有六个状态转移，分别为 S0->S1，S0->S3，S1->S2，S1->S0，S2->S0，S3->S0。

在本赛题中会关注其中更为重要的第二步，FSM Coverage 的计算。

VCD 文件完整的记录了仿真过程中信号变化的信息，可以通过 VCD 文件确定在仿真过程中，信号的取值变化，进而确定 FSM 的状态和状态转移是否被覆盖到。例如对于图 1 给出的 RTL 和 Test bench 示例，可以得到如下的 VCD 文件，如图 3 所示

```
$timescale
1ns
$end

$scope module test $end
$var reg 1 ! clk $end
$var reg 1 " reset $end
$var reg 1 # detect $end

$scope module test_fsm1 $end
$var wire 1 $ clk $end
$var wire 1 % reset $end
$var wire 1 & detect $end
$var reg 2 ' current [1:0] $end
$var reg 2 ( next [1:0] $end
$upscope $end

$upscope $end

$enddefinitions $end
```

```
#0
$dumpvars
0!
b00 (
bxx '
1&
0%
0$
1#
0"
$end
#50
1!
1$
b00 '
b01 (
#100
0!
0$
```

```
#150
b01 '
1$
1!
b10 (
#200
0$
0!
#250
b10 '
1!
b00 (
#300
0!
0$
#350
b00 '
1$
1!
b01 (
```

图 3 VCD 文件格式

上述 VCD 文件用可视化波形显示工具可以得到如图 4 的的波形显示。

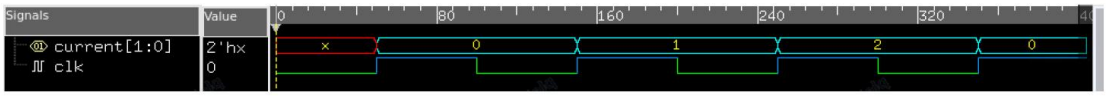


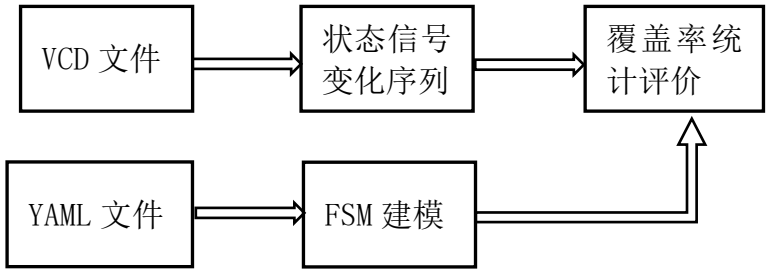

图 4 波形图

对照图 3 和图 4 的状态变量信号 current，可以看到状态信号 current 一共经历三种状态迁移，即 S0->S1、S1->S2、S2->S0，通过 YAML 文件的 FSM 分析可知，状态迁移一共有 6 中情况，因此此测试例的 FSM 覆盖率为 3/6=50%。

## 二、赛题要点

本赛题主要就是进行 YAML 文件的 FSM 建模，然后根据题目的要求，解析仿真 VCD 文件并分别得到针对单一 VCD 文件的 FSM 状态迁移统计（如状态变量信号 current，信号状态统计、信号状态的迁移统计等）、多个连续时间窗口的 FSM 状态迁移统计、多个 VCD 合并的 FSM 状态迁移统计，最后得到覆盖率统计结果。

因此，首先需要定义一些数据结构表示 FSM 中的状态及状态迁移关系，可以用数组或者链表等结构，先将 YAML 文件中的 FSM 进行建模；然后再针对将 VCD 文件读取、解析为状态信号时间序列变量，再针对这些变量进行统计分析，得到所需的输出。如下图所示：

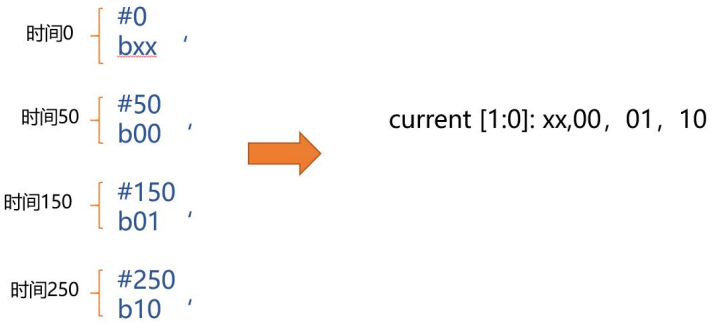



## 2.1 vcd 文件解析

vcd 文件解析的思路如下：

（1）首先扫描文件头，建立每个信号对应的数组或者链表结构。

（2）从 YAML 文件中提取 FSM 的状态变量，在 vcd 文件中扫描该信号变化的时间序列，对于一个给定时间段，扫描状态变量的值变化，记录其变化，就构成了状态变化序列。




这样，扫描完一个文件之后，各个变量随时间变化的序列就存下来了。

## 2.2 统计输出

有了状态变量信号的时间序列值，可以根据题目要求打印统计结果。主要考虑几方面：

（1）状态值迁移情况统计，需要统计不同的状态迁移，给出 FSM 迁移覆盖率。

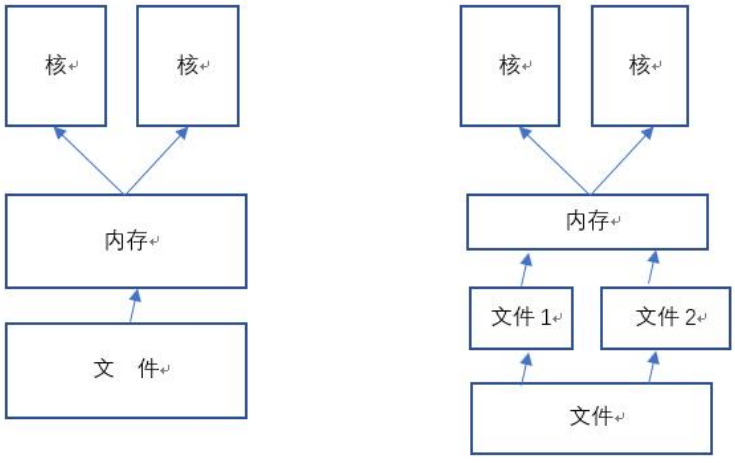
（2）指定多个时间窗口，分别进行状态迁移情况的统计，需要按照要求截取一个时间窗口进行计算。

（3）多场景仿真的信号，需要从相同 RTL 文件中的多个不同 VCD 文件中读取状态信号的迁移情况，合并计算覆盖率。

## 2.3 多核加速

在题目要求中提到了多核加速方法来提升分析统计的速度。在多核多线程的电脑上可以采用多线程编程的方式进行加速。统计任务划分为多个可并行执行的子任务，用多线程编程 pthread 或者 openMP 的方式同时进行计算统计，从而提升计算速度。

并行化方法包括有细粒度并行和粗粒度并行两种方式。细粒度并行化先将文件读入到内存，然后将不同的行分配到不同的核/线程来处理；粗粒度并行化则是先将大文件进行分割，然后分配到不同的处理器核，每个处理器核能处理不同段的文件。




使用并行化编程时需要注意线程间的同步和互锁机制的使用以保证数据读取的安全性。C/C ++语言会对并行化编程提供基于 Pthread 编程支持，编程效率

会更高；python 语言目前也能够提供多进程和多线程的并行化编程方式，但是 Python 的线程和进程的效率通常不如 C 程序中的线程和进程。

## 2.4 文件处理

对于大 vcd 文件，一次性读入会占用太多内存从而影响处理性能时，可以每次向内存中读入部分文件，例如 C/C++中采用 getline()等函数进行部分文件读入，可以每次读入一行文件。

### 三、解题步骤

（1）对 FSM 模型要先熟悉，能够准确的提取状态变量、状态类型以及总的状态迁移，得到静态 FSM 模型，作为覆盖率计算的基准；

（2）要先学习熟悉 VCD 的基本语法，如信号名字的对应关系，每一行表示什么含义，时间序列的表达方式等。

（3）再尝试实现从 VCD 文件中对状态变量时间序列的提取，这是一个文本逐行查找与解析的过程。并进一步完成统计分析模块的编写，每个模块对应不同的统计需求，相对独立。

（4）下一步再考虑 pthread 多线程编程，学习如何创建线程、线程间如何同步、加锁等。通过 pthread 提高程序效率的时候，需要文件进行合理划分后分配到不同的程序线程。

（5）尝试一下不同的文件读取速度和占用内存情况，选择一个占用内存比较小的方式。

### 四、其它注意事项

大家在解题时还需要注意以下几点：

1、不要轻易放弃，题目本身难度不大，但是前期的知识准备要花点时间；

2、赛题相关的知识并不难，但在缺乏相关背景的情况下需要克服一些困难。因此通过积极参加竞赛前的培训是一个很好的途径，请大家务必重视；

3、进行合理的问题分解，按照循序渐进、从易到难的解题步骤，逐步解答完成，切不可贪大求全。

最后，预祝各位同学发挥出自己的水平，取得良好的成绩。