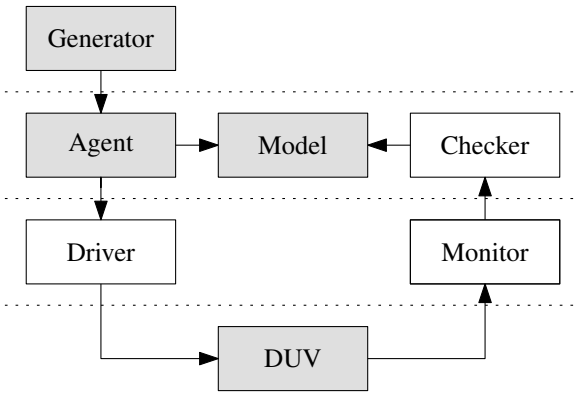
FH-OÖ Hagenberg/ESD
Advanced Methods of Verification, SS 2015
Rainer Findenig © 2008

5. Übung: Coverage

1 Code Coverage

Ein Test, der nur Teile des *Design under Verification* (DUV) verwendet bzw. abdeckt, kann offensichtlich auch nur in diesen Teilen Fehler finden. Über alle anderen Teile des DUV wird keine Aussage getroffen – sie können korrekt oder fehlerhaft sein, oder auch ganz fehlen. Daher ist der Grad der Abdeckung des Codes, die sogenannte *Code Coverage* , eine wichtige Metrik zur Bestimmung der Qualität der Testsuite:



Zusätzliche Bedeutung erhält die *Code Coverage* übrigens durch die Tatsache, dass sie durch das Simulationswerkzeug automatisch generiert wird, also bis auf Änderungen im Simulationsskript keinerlei Aufwand für den Entwickler entsteht. Diese Änderungen halten sich außerdem glücklicherweise auch in Grenzen (dargestellt für *QuestaSim*):

- die Quelldateien, für die *Coverage* -Informationen gesammelt werden sollen, müssen mit dem Befehl `vcom -coverAll myfile.vhd` kompiliert werden und
- die Simulation muss mit `vsim -coverage mytestbench` gestartet werden.

□ Beantworten Sie bitte die folgenden Fragen:

- Erklären Sie die Begriffe *Statement Coverage* , *Path Coverage* , *Expression Coverage* und *FSM Coverage* !
- Lässt sich in jedem Hardware-Entwurf eine *Code Coverage* von 100% erreichen? Begründen Sie Ihre Antwort!

- Bedeutet 100% *Code Coverage* , dass der getestete Entwurf fehlerfrei ist? Begründen Sie Ihre Antwort!

1.1 PROL16-Regression-Test

Instrumentieren Sie nun den im Elearning gegebenen Entwurf des PROL16 so, dass *Coverage* -Informationen erzeugt werden. Simulieren Sie danach den von Ihnen im fünften Semester erstellten Regressionstest und dokumentieren Sie die *Code-Coverage* -Ergebnisse (Befehl `coverage report`):

- Wo sehen Sie Lücken in der *Coverage* ? Wieso treten diese auf? Gibt es Code-Teile, die zwar in der ASE-Übung gefordert waren, aber nicht verwendet werden?
- Welche dieser Lücken können geschlossen werden, welche nicht?

2 Functional Coverage

Da die Aussagekraft der *Code Coverage* nur begrenzt ist, nimmt die *Functional Coverage* als zusätzliche Metrik einen sehr hohen Stellenwert in der Verifikation ein. Damit kann der Abdeckungsgrad von Funktionen des DUVs gemessen werden – Beispiele dazu wären etwa:

- Wie oft wurde der Befehl **ADDC** bei gesetztem Carry-Flag aufgerufen?
- Wurde durch den Befehl **SHL** jemals das Zero-Flag von 0 auf 1 gändert?
- Hat der Befehl **AND** jemals das Carry-Flag unverändert gelassen, wenn es zuvor 0 war?

Erweitern Sie Ihre SystemVerilog-Testbench nun so, dass (zumindest!) folgende Informationen gesammelt werden:

- Wie oft wurde jeder Befehl aufgerufen?
- Wie oft wurde jeder Befehl einem bestimmten Register als Ra/Rb aufgerufen?
- Wie oft wurde jeder Befehl mit gesetztem/gelöschtem Carry-/Zero-Flag aufgerufen?
- Wie oft hat jeder Befehl das Carry-/Zero-Flag von 0 auf 1 bzw. von 1 auf 0 geändert?
- Wie oft hat jeder Befehl das Carry-/Zero-Flag, wenn es zuvor 0 bzw. 1 war, unverändert gelassen?

Beachten Sie dabei, dass bestimmte Fälle nie auftreten können, weil sie durch die Spezifikation ausgeschlossen werden (**NOP** darf zum Beispiel nie ein Flag beeinflussen) – eine *Functional Coverage* von 100% wäre also mit einem korrekten Entwurf nicht erreichbar. Solche Fälle müssen also mit den *Coverage* -Regeln ausgeschlossen werden!

Hinweis: Um genauere Coverage-Informationen in QuestaSim zu sammeln, können Sie die `per_instance`-Option verwenden:

```
covergroup cg @(posedge clk);
    option.per_instance = 1;
    // ...
endgroup
```

"We didn't have to replicate the problem. We understood it."
Linus Torvalds