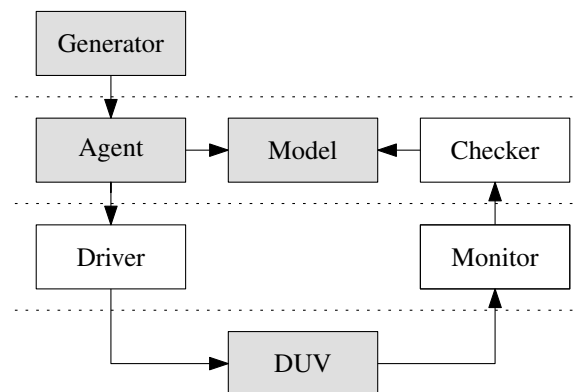


## 1 Code Coverage

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 geändert?
- 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:

```
1 covergroup cg @(posedge clk);  
2     option.per_instance = 1;  
3     // ...  
4 endgroup
```

*“We didn’t have to replicate the problem. We understood it.”*  
Linus Torvalds

# 1 Beispiel 1

## 1.1 Beantwortung der Fragen

Statement Coverage: Liefert Information darüber welche Zeilen/Statements des Codes ausgeführt wurden.

Path Coverage: Ist eine Erweiterung der Branch Coverage. Überprüft alle möglichen Kombinationen von Branches. Zum Beispiel bei zwei if-Anweisungen: true+true, false+true, true+false, false+false.

Expression Coverage: Überprüft die mögliche Werte bei einer Zuweisung und zählt mit wie oft welcher Wert zugewiesen wird. Bei einer logischen Zuweisung werden auch innere Ausdrücke betrachtet.

FSM Coverage: Gibt von einer FSM an wie oft jeder Zustand besucht wurde und wie oft jede einzelne Transition ausgeführt wurde.

Hardware-Entwurf: Nein, das Problem sind z.B. die default-Zweige. Diese können dann gegebenenfalls bei der Coverage ignoriert werden.

Code Coverage: Nein, Code Coverage heißt nur, dass der Code ausgeführt wurde, aber nicht, dass er fehlerfrei ist.

## 1.2 Coverage Report

Lücken:

- Register Index = -1 -> einfach Register > 32 nehmen
- Register Decode Error -> siehe Register Index
- Falscher Opcode -> kommt darauf an ob eine falscher Opcode in das Memory geladen werden

Coverage tcl-File:

```
../../sim/ComSimRegressionCoverage.tcl
```

```
1 vlib work
2 vcom -87 -coverAll ../src/vhdl/pro116_pack.vhd
3 vcom -87 -coverAll ../src/vhdl/reg_file.vhd
4 vcom -87 -coverAll ../src/vhdl/datapath.vhd
5 vcom -87 -coverAll ../src/vhdl/control.vhd
6 vcom -87 -coverAll ../src/vhdl/alu.vhd
7 vcom -93 -coverAll ../src/vhdl/memory.vhd
8 vcom -87 -coverAll ../src/vhdl/cpu.vhd
9 vcom -87 -coverAll ../src/vhdl/cpu_tb.vhd
10
11 vsim -gfile_base_g="regressiontest" -novopt -coverage cpu_tb
12 run 200 us
13
14 coverage report -file coverage.txt
```

Textdatei mit Coverage Report:

```
../../sim/coverage.txt
```

```
1 Coverage Report Summary Data by file
2
3 File: ../src/vhdl/alu.vhd
4     Enabled Coverage      Active      Hits      Misses % Covered
5     -----
```

6	Stmts	52	51	1	98.0
7	Branches	26	25	1	96.1
8	FEC Condition Terms	0	0	0	100.0
9	FEC Expression Terms	2	2	0	100.0
10	States	0	0	0	100.0
11	Transitions	0	0	0	100.0

12

13 File: ../src/vhdl/control.vhd

14	Enabled Coverage	Active	Hits	Misses	% Covered
15	-----	-----	----	-----	-----
16	Stmts	99	94	5	94.9
17	Branches	86	83	3	96.5
18	FEC Condition Terms	4	2	2	50.0
19	FEC Expression Terms	0	0	0	100.0
20	States	4	4	0	100.0
21	Transitions	12	7	5	58.3

22

23 File: ../src/vhdl/cpu.vhd

24	Enabled Coverage	Active	Hits	Misses	% Covered
25	-----	-----	----	-----	-----
26	Stmts	4	4	0	100.0
27	Branches	0	0	0	100.0
28	FEC Condition Terms	0	0	0	100.0
29	FEC Expression Terms	6	6	0	100.0
30	States	0	0	0	100.0
31	Transitions	0	0	0	100.0

32

33 File: ../src/vhdl/cpu\_tb.vhd

34	Enabled Coverage	Active	Hits	Misses	% Covered
35	-----	-----	----	-----	-----
36	Stmts	16	16	0	100.0
37	Branches	4	4	0	100.0
38	FEC Condition Terms	0	0	0	100.0
39	FEC Expression Terms	0	0	0	100.0
40	States	0	0	0	100.0
41	Transitions	0	0	0	100.0

42

43 File: ../src/vhdl/datapath.vhd

44	Enabled Coverage	Active	Hits	Misses	% Covered
45	-----	-----	----	-----	-----
46	Stmts	35	33	2	94.2
47	Branches	24	22	2	91.6
48	FEC Condition Terms	2	1	1	50.0
49	FEC Expression Terms	0	0	0	100.0
50	States	0	0	0	100.0
51	Transitions	0	0	0	100.0

52

53 File: ../src/vhdl/memory.vhd

54	Enabled Coverage	Active	Hits	Misses	% Covered
55	-----	-----	----	-----	-----
56	Stmts	799	501	298	62.7
57	Branches	417	237	180	56.8
58	FEC Condition Terms	143	43	100	30.0
59	FEC Expression Terms	6	1	5	16.6
60	States	0	0	0	100.0
61	Transitions	0	0	0	100.0

62

63 File: ../src/vhdl/reg\_file.vhd

64	Enabled Coverage	Active	Hits	Misses	% Covered
65	-----	-----	----	-----	-----

66	Stmts	3	3	0	100.0
67	Branches	9	8	1	88.8
68	FEC Condition Terms	0	0	0	100.0
69	FEC Expression Terms	0	0	0	100.0
70	States	0	0	0	100.0
71	Transitions	0	0	0	100.0

72

73

74 NEVER FAILED: 100.0% ASSERTIONS: 18

### Assemblercode:

../../sim/regressiontest.ass

```

1 ;; registers:
2 ;; r0      is always 0
3 ;; r1-r3   won't be changed during sub calls
4 ;; r4, r5 hold sub parameters
5 ;; r6, r7 are temp registers
6
7 equ prog_start, 0000h ; start address of program rom
8 equ data_start, 8000h ; start address of data ram
9 equ mem_dump_addr, 0ffffh
10
11
12 macro write_mem_dump
13     loadi r6, mem_dump_addr
14     store r4, r6           ; dump memory to file
15 endm
16
17 macro error_check
18     jumpz r3
19 error_loop:
20     loadi r5, error_loop
21     jump r5
22 endm
23
24 macro flags_check_1
25     loadi r4, check_z_1
26     jumpc r4
27 error_c_1_1:
28     loadi r5, error_c_1_1
29     jump r5
30 check_z_1:
31     loadi r4, end_1
32     jumpz r4
33 error_z_1_1:
34     loadi r5, error_z_1_1
35     jump r5
36 end_1:
37 endm
38
39 macro flags_check_0
40     loadi r4, error_0
41     jumpc r4
42 check_z_0:
43     loadi r4, error_0
44     jumpz r4
45     loadi r4, end_0
46     jump r4
47 error_0:
48     loadi r5, error_0

```

```

49     jump r5
50 end_0:
51 endm
52
53 macro z_flag_check_1
54     loadi r4, end_z_1
55     jumpz r4
56 error_z_1_2:
57     loadi r4, error_z_1_2
58     jump r4
59 end_z_1:
60 endm
61
62 macro c_flag_check_1
63     loadi r4, end_c_1
64     jumpc r4
65 error_c_1_2:
66     loadi r4, error_c_1_2
67     jump r4
68 end_c_1:
69 endm
70
71 macro z_flag_check_0
72     loadi r4, error_z_0
73     jumpz r4
74     loadi r4, end_z_0
75     jump r4
76 error_z_0:
77     loadi r5, error_0
78     jump r5
79 end_z_0:
80 endm
81
82 org data_start
83 x : db 10
84 store_x : db 0
85 max : db 65535
86 max_h : db 32768
87
88 org prog_start
89     loadi r0, 0
90
91 test_move:
92     loadi r6, x
93     load r1, r6
94     move r2, r1
95     ; error check
96     loadi r3, test_inc
97     ; loadi r2, 0 ; check the error check
98     comp r1, r2
99     error_check
100
101 test_inc:
102     loadi r6, max
103     load r1, r6
104     loadi r2, 1
105     add r2, r1 ; use add to compare
106     inc r1
107     flags_check_1
108     inc r1

```

```

109     inc r2
110     flags_check_0
111     ; error check
112     loadi r3, test_add
113     comp r1, r2
114     error_check
115
116 test_add:
117     loadi r6, max_h
118     load r1, r6
119     add r1, r1
120     flags_check_1
121     loadi r2, 1
122     add r1, r2
123     flags_check_0
124     ; error check
125     loadi r3, test_sub
126     comp r1, r2
127     error_check
128
129 test_sub:
130     loadi r1, 0
131     loadi r6, max
132     load r2, r6
133     sub r1, r2
134     c_flag_check_1
135     loadi r2, 1
136     sub r1, r2
137     z_flag_check_1
138     loadi r1, 2
139     loadi r2, 1
140     sub r1, r2
141     flags_check_0
142     ; error check
143     loadi r3, test_dec
144     comp r1, r2
145     error_check
146
147 test_dec:
148     loadi r1, 0
149     dec r1
150     c_flag_check_1
151     loadi r1, 1
152     dec r1
153     z_flag_check_1
154     loadi r1, 2
155     dec r1
156     flags_check_0
157     ; error check
158     loadi r3, test_addc
159     loadi r2, 1
160     comp r1, r2
161     error_check
162
163 test_addc:
164     loadi r6, max
165     load r1, r6
166     inc r1 ; overflow
167     loadi r6, max
168     load r1, r6

```



```

169     addc r1, r0
170     flags_check_1
171     loadi r1, 1
172     addc r1, r1
173     flags_check_0
174     loadi r6, max
175     load r1, r6
176     inc r1 ; overflow
177     loadi r1, 1
178     addc r1, r1
179     loadi r2, 3
180     ; error check
181     loadi r3, test_subc
182     comp r1, r2
183     error_check
184
185 test_subc:
186     loadi r6, max
187     load r1, r6
188     inc r1 ; overflow
189     loadi r1, 0
190     loadi r6, max
191     subc r1, r2
192     ;flags_check_1
193     c_flag_check_1
194     loadi r1, 3
195     loadi r2, 1
196     subc r1, r2
197     flags_check_0
198     loadi r6, max
199     load r1, r6
200     inc r1 ; overflow
201     loadi r1, 3
202     loadi r2, 1
203     subc r1, r2
204     ; error check
205     loadi r3, test_comp
206     loadi r2, 1
207     comp r1, r2
208     error_check
209
210 test_comp:
211     loadi r1, 1
212     comp r1, r1
213     z_flag_check_1
214     loadi r2, 2
215     comp r1, r2
216     z_flag_check_0
217
218 test_and:
219     loadi r1, 0
220     and r1, r1
221     z_flag_check_1
222     loadi r1, 1
223     and r1, r1
224     flags_check_0
225     and r1, r0
226     ; error check
227     loadi r3, test_or
228     comp r1, r0

```

```

229     error_check
230
231 test_or:
232     loadl r1, 0
233     or r1, r1
234     z_flag_check_1
235     loadl r1, 1
236     or r1, r0
237     flags_check_0
238     ; error check
239     loadl r3, test_xor
240     loadl r2, 1
241     comp r1, r2
242     error_check
243
244 test_xor:
245     loadl r1, 0
246     xor r1, r1
247     z_flag_check_1
248     loadl r2, 1
249     xor r1, r2
250     flags_check_0
251     ; error check
252     loadl r3, test_not
253     comp r1, r2
254     error_check
255
256 test_not:
257     move r1, r0
258     not r1
259     flags_check_0
260     not r1
261     z_flag_check_1
262
263 test_shl:
264     loadl r6, max_h
265     load r1, r6
266     shl r1
267     flags_check_1
268     loadl r1, 1
269     shl r1
270     flags_check_0
271
272 test_shr:
273     loadl r1, 1
274     shr r1
275     flags_check_1
276     loadl r6, max_h
277     load r1, r6
278     shr r1
279     flags_check_0
280
281 test_shlc:
282     add r0, r0 ; carry = 0
283     loadl r6, max_h
284     load r1, r6
285     shlc r1
286     flags_check_1
287     loadl r6, max
288     load r1, r6

```



```

10     logic cpu_halt;
11
12     modport master (
13         output mem_data_tb,
14         input  mem_data_cpu, mem_ce_n, mem_oe_n, illegal_inst, cpu_halt
15     );
16
17 endinterface

```

../../src/sv/pkgProl16.sv

```

1 package pkgProl16;
2     typedef bit [15:0] data_v;
3 endpackage

```

../../src/sv/Prol16Command.sv

```

1 typedef enum bit [5:0]
2     {
3         Nop=6'b000000,
4         Loadi=6'b000010,
5         Jump=6'b001000,
6         Jumpc=6'b001010,
7         Jumpz=6'b001011,
8         Move=6'b001100,
9         And=6'b010000,
10        Or=6'b010001,
11        Xor=6'b010010,
12        Not=6'b010011,
13        Add=6'b010100,
14        Addc=6'b010101,
15        Sub=6'b010110,
16        Subc=6'b010111,
17        Comp=6'b011000,
18        Inc=6'b011010,
19        Dec=6'b011011,
20        Shl=6'b011100,
21        Shr=6'b011101,
22        Shlc=6'b011110,
23        Shrc=6'b011111
24        //Invalid=6'b111111
25    } Prol16Command;

```

../../src/sv/Prol16Opcode.sv

```

1 `include "Prol16Command.sv"
2
3 class Prol16Opcode;
4     rand int ra;
5     rand int rb;
6     randc Prol16Command cmd;
7     rand pkgProl16::data_v data;
8
9     constraint ra_size {ra >= 0 && ra <= 31};
10    constraint rb_size {rb >= 0 && rb <= 31};
11
12    function new(int ra, int rb, Prol16Command cmd, pkgProl16::data_v data);
13        this.ra = ra;
14        this.rb = rb;
15        this.cmd = cmd;
16        this.data = data;
17    endfunction
18 endclass

```

../../src/sv/Prol16State.sv

```
1 class Prol16State #(parameter int gRegs = 32);
2     pkgProl16::data_v regs[gRegs];
3     pkgProl16::data_v programCounter;
4     bit cFlag;
5     bit zFlag;
6
7     function new();
8         for (int i = 0; i < gRegs; i++) begin
9             regs[i] = '0;
10        end
11        programCounter = 0;
12        cFlag = 0;
13        zFlag = 0;
14    endfunction
15
16    task reset();
17        for (int i = 0; i < gRegs; i++) begin
18            regs[i] = '0;
19        end
20        programCounter = 0;
21        cFlag = 0;
22        zFlag = 0;
23    endtask
24 endclass
```

../../src/sv/Prol16Model.sv

```
1 `include "Prol16State.sv"
2 `include "Prol16Opcode.sv"
3
4 class Prol16Model#(parameter int gNumRegs);
5     Prol16State state;
6
7     function new(Prol16State _state);
8         state = _state;
9     endfunction
10
11    task reset();
12        state.reset();
13    endtask
14
15    task calcCarryFlag(int data);
16        if (data > (2**$size(pkgProl16::data_v) - 1) || data < 0)
17            state.cFlag = 1;
18        else
19            state.cFlag = 0;
20    endtask
21
22    task calcZeroFlag(pkgProl16::data_v data);
23        if (data == '0)
24            state.zFlag = 1;
25        else
26            state.zFlag = 0;
27    endtask
28
29    task execute(Prol16Opcode opcode);
30        int data = 0; // tmp data for carry calc
31        bit tmpCFlag = 0; // tmp carry flag
32
33        // inc pc here, overwrite in jumps
```

```

34     state.programCounter++;
35
36     if (opcode.ra > (gNumRegs - 1) || opcode.ra < 0 ||
37         opcode.rb > (gNumRegs - 1) || opcode.rb < 0)
38         $display("Illegal register, doing nop");
39     else
40         begin
41
42             case (opcode.cmd)
43             Nop:
44                 begin
45                     end
46
47             Loadi:
48                 begin
49                     state.regs[opcode.ra] = opcode.data;
50                     state.programCounter++; // TODO: necessary?
51                 end
52
53             Jump:
54                 begin
55                     state.programCounter = state.regs[opcode.ra];
56                 end
57
58             Jumpc:
59                 begin
60                     if (state.cFlag == 1)
61                         state.programCounter = state.regs[opcode.ra];
62                 end
63
64             Jumpz:
65                 begin
66                     if (state.zFlag == 1)
67                         state.programCounter = state.regs[opcode.ra];
68                 end
69
70             Move:
71                 begin
72                     state.regs[opcode.ra] = state.regs[opcode.rb];
73                 end
74
75             And:
76                 begin
77                     state.regs[opcode.ra] = state.regs[opcode.ra] & state.regs[
78                         opcode.rb];
79                     state.cFlag = 0;
80                     calcZeroFlag(state.regs[opcode.ra]);
81                 end
82
83             Or:
84                 begin
85                     state.regs[opcode.ra] = state.regs[opcode.ra] | state.regs[
86                         opcode.rb];
87                     state.cFlag = 0;
88                     calcZeroFlag(state.regs[opcode.ra]);
89                 end
90
91             Xor:
92                 begin

```

```

91         state.regs[opcode.ra] = state.regs[opcode.ra] ^ state.regs[
           opcode.rb];
92         state.cFlag = 0;
93         calcZeroFlag(state.regs[opcode.ra]);
94     end
95
96     Not:
97     begin
98         state.regs[opcode.ra] = ~(state.regs[opcode.ra]);
99         state.cFlag = 0;
100        calcZeroFlag(state.regs[opcode.ra]);
101    end
102
103     Add:
104     begin
105         data = state.regs[opcode.ra] + state.regs[opcode.rb];
106         calcCarryFlag(data);
107         state.regs[opcode.ra] = data;
108         calcZeroFlag(state.regs[opcode.ra]);
109     end
110
111     Addc:
112     begin
113         data = state.regs[opcode.ra] + state.regs[opcode.rb] + state.
           cFlag;
114         calcCarryFlag(data);
115         state.regs[opcode.ra] = data;
116         calcZeroFlag(state.regs[opcode.ra]);
117     end
118
119     Sub:
120     begin
121         data = state.regs[opcode.ra] - state.regs[opcode.rb];
122         calcCarryFlag(data);
123         state.regs[opcode.ra] = data;
124         calcZeroFlag(state.regs[opcode.ra]);
125     end
126
127     Subc:
128     begin
129         data = state.regs[opcode.ra] - state.regs[opcode.rb] - state.
           cFlag;
130         calcCarryFlag(data);
131         state.regs[opcode.ra] = data;
132         calcZeroFlag(state.regs[opcode.ra]);
133     end
134
135     Comp:
136     begin
137         data = state.regs[opcode.ra] - state.regs[opcode.rb];
138         calcCarryFlag(data);
139         calcZeroFlag(data);
140     end
141
142     Inc:
143     begin
144         data = state.regs[opcode.ra] + 1;
145         calcCarryFlag(data);
146         state.regs[opcode.ra] = data;
147         calcZeroFlag(state.regs[opcode.ra]);

```

```

148     end
149
150     Dec:
151     begin
152         data = state.regs[opcode.ra] - 1;
153         calcCarryFlag(data);
154         state.regs[opcode.ra] = data;
155         calcZeroFlag(state.regs[opcode.ra]);
156     end
157
158     Shl:
159     begin
160         if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
161             state.cFlag = 1;
162         else
163             state.cFlag = 0;
164
165             state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
166             calcZeroFlag(state.regs[opcode.ra]);
167     end
168
169     Shr:
170     begin
171         if (state.regs[opcode.ra][0] == 1)
172             state.cFlag = 1;
173         else
174             state.cFlag = 0;
175
176             state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
177             calcZeroFlag(state.regs[opcode.ra]);
178     end
179
180     Shlc:
181     begin
182         if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
183             tmpCFlag = 1;
184         else
185             tmpCFlag = 0;
186
187             state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
188             state.regs[opcode.ra][0] = state.cFlag;
189             state.cFlag = tmpCFlag;
190             calcZeroFlag(state.regs[opcode.ra]);
191     end
192
193     Shrc:
194     begin
195         if (state.regs[opcode.ra][0] == 1)
196             tmpCFlag = 1;
197         else
198             tmpCFlag = 0;
199
200             state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
201             state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] = state.
                cFlag;
202             state.cFlag = tmpCFlag;
203             calcZeroFlag(state.regs[opcode.ra]);
204     end
205
206     default : $display("Wrong opcode: doing nop");

```



```

207         endcase
208     end
209 endtask
210 endclass

```

../src/sv/testProl16Rand.sv

```

1  `include "Prol16Model.sv"
2
3  class TestClass;
4      task assertWithoutFlags(int expectedRegisterValue, int expectedPCValue,
        Prol16State state, int register, string text);
5          assert (expectedRegisterValue == state.regs[register])
6          else $error("Expected Register Value: %d, Actual Register Value: %d,
            Info: %s", expectedRegisterValue, state.regs[register], text);
7
8          assert (expectedPCValue == state.programCounter)
9          else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
            expectedPCValue, state.programCounter, text);
10     endtask
11
12     task assertWithFlags(int expectedRegisterValue, int expectedPCValue, int
        expectedCarryFlag, int expectedZeroFlag, Prol16State state, int
        register, string text);
13         assert (expectedRegisterValue == state.regs[register])
14         else $error("Expected Register Value: %d, Actual Register Value: %d,
            Info: %s", expectedRegisterValue, state.regs[register], text);
15
16         assert (expectedPCValue == state.programCounter)
17         else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
            expectedPCValue, state.programCounter, text);
18
19         assert (expectedCarryFlag == state.cFlag)
20         else $error("Expected Carry Flag Value: %d, Actual Carry Flag Value:
            %d, Info: %s", expectedCarryFlag, state.cFlag, text);
21
22         assert (expectedZeroFlag == state.zFlag)
23         else $error("Expected Zero Flag Value: %d, Actual Zero Flag Value: %d
            , Info: %s", expectedZeroFlag, state.zFlag, text);
24     endtask
25
26     task assertWithDuv(Prol16State state, int register, logic [15:0] cpuRegs
        [31:0], logic[15:0] cpuPc, logic cpuCFlag, logic cpuZFlag,
        Prol16Command cmd);
27         assert (state.regs[register] == cpuRegs[register])
28         else $error("WithDuv: Expected Register Value: %d, Actual Register
            Value: %d, Info: %s", state.regs[register], cpuRegs[register], cmd
            .name());
29
30         assert (state.programCounter == cpuPc)
31         else $error("WithDuv: Expected PC Value: %d, Actual PC Value: %d,
            Info: %s", state.programCounter, cpuPc, cmd.name());
32
33         assert (state.cFlag == cpuCFlag)
34         else $error("WithDuv: Expected Carry Flag Value: %d, Actual Carry
            Flag Value: %d, Info: %s", state.cFlag, cpuCFlag, cmd.name());
35
36         assert (state.zFlag == cpuZFlag)
37         else $error("WithDuv: Expected Zero Flag Value: %d, Actual Zero Flag
            Value: %d, Info: %s", state.zFlag, cpuZFlag, cmd.name());
38     endtask

```

```

39 endclass
40
41 program testProll16Rand(iffProll16.master cpu, output logic rst, input logic
    clk);
42     logic [15:0] cpuRegs [31:0];
43     logic [15:0] cpuPc;
44     logic cpuCFlag;
45     logic cpuZFlag;
46     logic [5:0] opcodeDUV;
47     logic [4:0] ra;
48     logic [4:0] rb;
49
50     logic [5:0] prevOpcodeDUV;
51
52     const int numTestCases = 100000;
53
54     event CommandStart;
55     event End;
56
57     bit LoadiOccurred = 0;
58
59     Proll16Opcode opcode = new(0, 0, Nop, 0);
60
61     task trigger();
62         while (!End.triggered) begin
63             @(negedge cpu.mem_oe_n)
64             begin
65                 //$display("negEdge oe");
66                 if (LoadiOccurred == 0)
67                     begin
68                         cpu.mem_data_tb[15:10] <= opcode.cmd;
69                         cpu.mem_data_tb[9:5] <= opcode.ra;
70                         cpu.mem_data_tb[4:0] <= opcode.rb;
71
72                         prevOpcodeDUV <= opcodeDUV;
73
74                         if (opcode.cmd == Loadi)
75                             begin
76                                 LoadiOccurred = 1;
77                             end
78                         end
79                     else
80                     begin
81                         cpu.mem_data_tb <= opcode.data;
82                         LoadiOccurred = 0;
83                     end
84                 end
85             @(posedge cpu.mem_oe_n)
86             begin
87                 //$display("posEdge oe");
88                 -> CommandStart;
89             end
90         end
91     endtask
92
93     //always_ff @(negedge clk) begin
94     // previous_opcode <= opcode;
95     //end
96
97     covergroup CoverCpu @(negedge cpu.mem_oe_n); // TODO, the right signal?

```

```

98     option.per_instance = 1;
99
100    cmd: coverpoint opcodeDUV iff (rst) {
101        bins NOP      = {0};
102        ignore_bins SLEEP = {1};
103        bins LOADI    = {2};
104        ignore_bins LOAD  = {3};
105        ignore_bins STORE = {4};
106        bins JUMP      = {8};
107        bins JUMPC     = {10};
108        bins JUMPZ     = {11};
109        bins MOVE      = {12};
110        bins AND       = {16};
111        bins OR        = {17};
112        bins XOR       = {18};
113        bins NOT       = {19};
114        bins ADD       = {20};
115        bins ADDC      = {21};
116        bins SUB       = {22};
117        bins SUBC      = {23};
118        bins COMP      = {24};
119        bins INC       = {26};
120        bins DEC       = {27};
121        bins SHL       = {28};
122        bins SHR       = {29};
123        bins SHLC      = {30};
124        bins SHRC      = {31};
125        bins Invalid   = {63};
126        illegal_bins other = default;
127    }
128
129    prevCmd: coverpoint prevOpcodeDUV iff (rst) {
130        bins NOP      = {0};
131        ignore_bins SLEEP = {1};
132        bins LOADI    = {2};
133        ignore_bins LOAD  = {3};
134        ignore_bins STORE = {4};
135        bins JUMP      = {8};
136        bins JUMPC     = {10};
137        bins JUMPZ     = {11};
138        bins MOVE      = {12};
139        bins AND       = {16};
140        bins OR        = {17};
141        bins XOR       = {18};
142        bins NOT       = {19};
143        bins ADD       = {20};
144        bins ADDC      = {21};
145        bins SUB       = {22};
146        bins SUBC      = {23};
147        bins COMP      = {24};
148        bins INC       = {26};
149        bins DEC       = {27};
150        bins SHL       = {28};
151        bins SHR       = {29};
152        bins SHLC      = {30};
153        bins SHRC      = {31};
154        bins Invalid   = {63};
155        //illegal_bins other = default;
156    }
157

```

```

158     ra: coverpoint ra;
159     rb: coverpoint rb;
160
161     crs_cmd_ra: cross cmd, ra;
162     crs_cmd_rb: cross cmd, rb;
163
164     cFlag: coverpoint cpuCFlag {
165         bins set = {1};
166         bins notset = {0};
167     }
168
169     zFlag: coverpoint cpuZFlag {
170         bins set = {1};
171         bins notset = {0};
172     }
173
174     crs_cmd_c: cross cmd, cFlag;
175     crs_cmd_z: cross cmd, zFlag;
176
177     trans_c: coverpoint cpuCFlag {
178         bins from_0_to_1 = (0 ==> 1);
179         bins from_1_to_0 = (1 ==> 0);
180         bins from_0_to_0 = (0 ==> 0);
181         bins from_1_to_1 = (1 ==> 1);
182     }
183
184     trans_z: coverpoint cpuZFlag {
185         bins from_0_to_1 = (0 ==> 1);
186         bins from_1_to_0 = (1 ==> 0);
187         bins from_0_to_0 = (0 ==> 0);
188         bins from_1_to_1 = (1 ==> 1);
189     }
190
191     prevCmdChangeFlags: coverpoint prevOpcodeDUV {
192         bins doesnt_change_flags = { 0, 1, 2, 3, 4, 8, 10, 11, 12};
193         bins c_flag_0_and_z_flag_x = { 16, 17, 18, 19 };
194         bins c_flag_x_and_z_flag_x = { 20, 21, 22, 23, 24, 26, 27, 28, 29,
195             30, 31 };
196     }
197
198     crs_cmd_c_change: cross prevCmdChangeFlags, trans_c {
199         illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
200             (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_0));
201     }
202
203     crs_cmd_z_change: cross prevCmdChangeFlags, trans_z {
204         illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
205             (binsof(trans_z.from_0_to_1) || binsof(trans_z.from_1_to_0));
206     }
207
208     crs_cmd_c_is_0: cross prevCmdChangeFlags, trans_c {
209         illegal_bins ill = binsof(prevCmdChangeFlags.c_flag_0_and_z_flag_x)
210             && (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_1));
211     }
212
213 endgroup
214
215 initial begin : stimuli
216     Prol16State state = new();

```

```

214     Prol16Model#(32) model = new(state);
215
216     TestClass testClass = new();
217
218     CoverCpu coverCpu = new;
219
220     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(0)"
221         , "/top/TheTest/cpuRegs(0)");
222     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(1)"
223         , "/top/TheTest/cpuRegs(1)");
224     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(2)"
225         , "/top/TheTest/cpuRegs(2)");
226     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(3)"
227         , "/top/TheTest/cpuRegs(3)");
228     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(4)"
229         , "/top/TheTest/cpuRegs(4)");
230     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(5)"
231         , "/top/TheTest/cpuRegs(5)");
232     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(6)"
233         , "/top/TheTest/cpuRegs(6)");
234     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(7)"
235         , "/top/TheTest/cpuRegs(7)");
236     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(8)"
237         , "/top/TheTest/cpuRegs(8)");
238     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(9)"
239         , "/top/TheTest/cpuRegs(9)");
240     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(10)"
241         , "/top/TheTest/cpuRegs(10)");
242     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(11)"
243         , "/top/TheTest/cpuRegs(11)");
244     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(12)"
245         , "/top/TheTest/cpuRegs(12)");
246     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(13)"
247         , "/top/TheTest/cpuRegs(13)");
248     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(14)"
249         , "/top/TheTest/cpuRegs(14)");
250     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(15)"
251         , "/top/TheTest/cpuRegs(15)");
252     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(16)"
253         , "/top/TheTest/cpuRegs(16)");
254     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(17)"
255         , "/top/TheTest/cpuRegs(17)");
256     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(18)"
257         , "/top/TheTest/cpuRegs(18)");
258     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(19)"
259         , "/top/TheTest/cpuRegs(19)");
260     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(20)"
261         , "/top/TheTest/cpuRegs(20)");
262     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(21)"
263         , "/top/TheTest/cpuRegs(21)");
264     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(22)"
265         , "/top/TheTest/cpuRegs(22)");
266     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(23)"
267         , "/top/TheTest/cpuRegs(23)");
268     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(24)"
269         , "/top/TheTest/cpuRegs(24)");
270     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(25)"
271         , "/top/TheTest/cpuRegs(25)");
272     $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(26)"
273         , "/top/TheTest/cpuRegs(26)");

```

```

247 $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(27)
    ", "/top/TheTest/cpuRegs(27)");
248 $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(28)
    ", "/top/TheTest/cpuRegs(28)");
249 $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(29)
    ", "/top/TheTest/cpuRegs(29)");
250 $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(30)
    ", "/top/TheTest/cpuRegs(30)");
251 $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(31)
    ", "/top/TheTest/cpuRegs(31)");
252
253 $init_signal_spy("/top/TheCpu/datapath_inst/RegPC", "/top/TheTest/
    cpuPc");
254 $init_signal_spy("/top/TheCpu/control_inst/Carry", "/top/TheTest/
    cpuCFlag");
255 $init_signal_spy("/top/TheCpu/control_inst/Zero", "/top/TheTest/
    cpuZFlag");
256 $init_signal_spy("/top/TheCpu/datapath_inst/RegOpcode", "/top/TheTest
    /opcodeDUV");
257 $init_signal_spy("/top/TheCpu/datapath_inst/RegAIdx", "/top/TheTest/
    ra");
258 $init_signal_spy("/top/TheCpu/datapath_inst/RegBIdx", "/top/TheTest/
    rb");
259
260 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(0)", "
    16#0000", 0, 1);
261 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(1)", "
    16#0000", 0, 1);
262 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(2)", "
    16#0000", 0, 1);
263 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(3)", "
    16#0000", 0, 1);
264 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(4)", "
    16#0000", 0, 1);
265 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(5)", "
    16#0000", 0, 1);
266 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(6)", "
    16#0000", 0, 1);
267 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(7)", "
    16#0000", 0, 1);
268 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(8)", "
    16#0000", 0, 1);
269 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(9)", "
    16#0000", 0, 1);
270 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(10)",
    "16#0000", 0, 1);
271 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(11)",
    "16#0000", 0, 1);
272 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(12)",
    "16#0000", 0, 1);
273 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(13)",
    "16#0000", 0, 1);
274 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(14)",
    "16#0000", 0, 1);
275 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(15)",
    "16#0000", 0, 1);
276 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(16)",
    "16#0000", 0, 1);
277 $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(17)",
    "16#0000", 0, 1);

```

```

278     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(18)",
279         "16#0000", 0, 1);
280     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(19)",
281         "16#0000", 0, 1);
282     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(20)",
283         "16#0000", 0, 1);
284     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(21)",
285         "16#0000", 0, 1);
286     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(22)",
287         "16#0000", 0, 1);
288     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(23)",
289         "16#0000", 0, 1);
290     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(24)",
291         "16#0000", 0, 1);
292     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(25)",
293         "16#0000", 0, 1);
294     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(26)",
295         "16#0000", 0, 1);
296     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(27)",
297         "16#0000", 0, 1);
298     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(28)",
299         "16#0000", 0, 1);
300     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(29)",
301         "16#0000", 0, 1);
302     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(30)",
303         "16#0000", 0, 1);
304     $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(31)",
305         "16#0000", 0, 1);
306
307     // generate reset
308     -----
309     rst = 1;
310     #10 rst = 0;
311     #20 rst = 1;
312
313     //Reset
314     model.reset();
315     //testClass.assertWithoutFlags(0, 0, model.state, 12, "Reset test");
316
317     fork
318         trigger();
319     join_none
320
321     //Nop
322     @(CommandStart);
323     testClass.assertWithDuv(model.state, 12, cpuRegs, cpuPc, cpuCFlag,
324         cpuZFlag, opcode.cmd);
325
326     for (int i = 0; i < numTestCases; i++) begin
327         if (opcode.cmd == Loadi)
328             begin
329                 @(CommandStart);
330             end
331         model.execute(opcode);
332         opcode.randomize();
333         //$display("Command %b", opcode.cmd);
334         //$display("CommandDUV %b", opcode.DUV);
335         @(CommandStart);

```

```

322         testClass.assertWithDuv(model.state, opcode.ra, cpuRegs, cpuPc,
           cpuCFlag, cpuZFlag, opcode.cmd);
323     end
324
325     -> End;
326     $stop;
327     end : stimuli
328 endprogram

```

../src/sv/top.sv

```

1  `include "testProl16Rand.sv"
2
3  module top;
4      logic clk = 0, rst;
5      always #10 clk = ~clk;
6
7      ifProl16 ifCpu();
8      cpu TheCpu(clk, rst, ifCpu.mem_addr, ifCpu.mem_data_cpu, ifCpu.
           mem_data_tb, ifCpu.mem_ce_n, ifCpu.mem_oe_n,
9           ifCpu.mem_we_n, ifCpu.illegal_inst, ifCpu.cpu_halt);
10     testProl16Rand TheTest(ifCpu.master, rst, clk);
11 endmodule

```

../sim/CompileSimRandCov.do

```

1  vlib work
2
3  vlog ../src/sv/ifProl16.sv
4  vlog ../src/sv/pkgProl16.sv
5  vlog ../src/sv/Prol16Command.sv
6  vlog ../src/sv/Prol16Opcode.sv +incdir+../src/sv
7  vlog ../src/sv/Prol16State.sv +incdir+../src/sv
8  vlog ../src/sv/Prol16Model.sv +incdir+../src/sv
9  vlog ../src/sv/testProl16Rand.sv +incdir+../src/sv
10 vlog ../src/sv/top.sv +incdir+../src/sv
11
12 vcom -87 ../src/vhdl/prol16_pack.vhd
13 vcom -87 ../src/vhdl/reg_file.vhd
14 vcom -87 ../src/vhdl/datapath.vhd
15 vcom -87 ../src/vhdl/control.vhd
16 vcom -87 ../src/vhdl/alu.vhd
17 vcom -93 ../src/vhdl/memory.vhd
18 vcom -87 ../src/vhdl/cpu.vhd
19 #vcom -87 ../src/vhdl/cpu_tb.vhd
20
21 vsim -novopt top
22 run -all

```