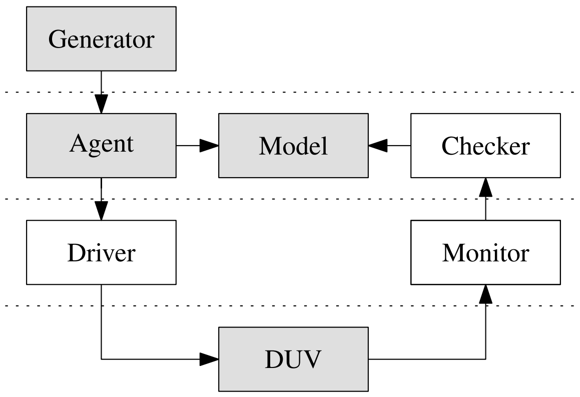
FH-OÖ Hagenberg/ESD
Advanced Methods of Verification, SS 2015
Rainer Findenig © 2008

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

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 einzele 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

```
vlib work
vcom -87 -coverAll ../src/vhdl/prol16_pack.vhd
vcom -87 -coverAll ../src/vhdl/reg_file.vhd
vcom -87 -coverAll ../src/vhdl/datapath.vhd
vcom -87 -coverAll ../src/vhdl/control.vhd
vcom -87 -coverAll ../src/vhdl/alu.vhd
vcom -93 -coverAll ../src/vhdl/memory.vhd
vcom -87 -coverAll ../src/vhdl/cpu.vhd
vcom -87 -coverAll ../src/vhdl/cpu_tb.vhd

vsim -gfile_base_g="regressiontest" -novopt -coverage cpu_tb
run 200 us

coverage report -file coverage.txt
```

Textdatei mit Coverage Report:

../../sim/coverage.txt

```
Coverage Report Summary Data by file

File: ../src/vhdl/alu.vhd
    Enabled Coverage        Active      Hits    Misses % Covered
    ----------------        ------      ----    ------ ---------
    Stmts                       52         51          1      98.0
    Branches                    26         25          1      96.1
    FEC Condition Terms          0          0          0     100.0
    FEC Expression Terms         2          2          0     100.0
    States                       0          0          0     100.0
    Transitions                  0          0          0     100.0

File: ../src/vhdl/control.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                       99         94          5      94.9
    Branches                    86         83          3      96.5
    FEC Condition Terms          4          2          2      50.0
    FEC Expression Terms         0          0          0     100.0
    States                       4          4          0     100.0
    Transitions                 12          7          5      58.3

File: ../src/vhdl/cpu.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                        4          4          0     100.0
    Branches                     0          0          0     100.0
    FEC Condition Terms          0          0          0     100.0
    FEC Expression Terms         6          6          0     100.0
    States                       0          0          0     100.0
    Transitions                  0          0          0     100.0

File: ../src/vhdl/cpu_tb.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                       16         16          0     100.0
    Branches                     4          4          0     100.0
    FEC Condition Terms          0          0          0     100.0
    FEC Expression Terms         0          0          0     100.0
    States                       0          0          0     100.0
    Transitions                  0          0          0     100.0

File: ../src/vhdl/datapath.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                       35         33          2      94.2
    Branches                    24         22          2      91.6
    FEC Condition Terms          2          1          1      50.0
    FEC Expression Terms         0          0          0     100.0
    States                       0          0          0     100.0
    Transitions                  0          0          0     100.0

File: ../src/vhdl/memory.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                      799        501        298      62.7
    Branches                   417        237        180      56.8
    FEC Condition Terms        143         43        100      30.0
    FEC Expression Terms         6          1          5      16.6
    States                       0          0          0     100.0
    Transitions                  0          0          0     100.0

File: ../src/vhdl/reg_file.vhd
    Enabled Coverage        Active       Hits     Misses % Covered
    ----------------        ------       ----     ------ ---------
    Stmts                         3         3         0     100.0
    Branches                      9         8         1      88.8
    FEC Condition Terms           0         0         0     100.0
    FEC Expression Terms          0         0         0     100.0
    States                        0         0         0     100.0
    Transitions                   0         0         0     100.0


NEVER FAILED: 100.0%  ASSERTIONS: 18
```

Assemblercode:

../../sim/regressiontest.ass

```
;; registers:
;; r0     is always 0
;; r1-r3  won't be changed during sub calls
;; r4, r5 hold sub parameters
;; r6, r7 are temp registers

equ prog_start, 0000h ; start address of program rom
equ data_start, 8000h ; start address of data ram
equ mem_dump_addr, 0ffffh


macro write_mem_dump
  loadi r6, mem_dump_addr
  store r4, r6            ; dump memory to file
endm

macro error_check
  jumpz r3
error_loop:
  loadi r5, error_loop
  jump r5
endm

macro flags_check_1
  loadi r4, check_z_1
  jumpc r4
error_c_1_1:
  loadi r5, error_c_1_1
  jump r5
check_z_1:
  loadi r4, end_1
  jumpz r4
error_z_1_1:
  loadi r5, error_z_1_1
  jump r5
end_1:
endm

macro flags_check_0
  loadi r4, error_0
  jumpc r4
check_z_0:
  loadi r4, error_0
  jumpz r4
  loadi r4, end_0
  jump r4
error_0:
  loadi r5, error_0
  jump r5
end_0:
endm

macro z_flag_check_1
  loadi r4, end_z_1
  jumpz r4
error_z_1_2:
  loadi r4, error_z_1_2
  jump r4
end_z_1:
endm

macro c_flag_check_1
  loadi r4, end_c_1
  jumpc r4
error_c_1_2:
  loadi r4, error_c_1_2
  jump r4
end_c_1:
endm

macro z_flag_check_0
  loadi r4, error_z_0
  jumpz r4
  loadi r4, end_z_0
  jump r4
error_z_0:
  loadi r5, error_0
  jump r5
end_z_0:
endm

org data_start
x : db 10
store_x : db 0
max : db 65535
max_h : db 32768

org prog_start
  loadi r0, 0

test_move:
  loadi r6, x
  load r1, r6
  move r2, r1
  ; error check
  loadi r3, test_inc
  ; loadi r2, 0 ; check the error check
  comp r1, r2
  error_check

test_inc:
  loadi r6, max
  load r1, r6
  loadi r2, 1
  add r2, r1 ; use add to compare
  inc r1
  flags_check_1
  inc r1
  inc r2
  flags_check_0
  ; error check
  loadi r3, test_add
  comp r1, r2
  error_check

test_add:
  loadi r6, max_h
  load r1, r6
  add r1, r1
  flags_check_1
  loadi r2, 1
  add r1, r2
  flags_check_0
  ; error check
  loadi r3, test_sub
  comp r1, r2
  error_check

test_sub:
  loadi r1, 0
  loadi r6, max
  load r2, r6
  sub r1, r2
  c_flag_check_1
  loadi r2, 1
  sub r1, r2
  z_flag_check_1
  loadi r1, 2
  loadi r2, 1
  sub r1, r2
  flags_check_0
  ; error check
  loadi r3, test_dec
  comp r1, r2
  error_check

test_dec:
  loadi r1, 0
  dec r1
  c_flag_check_1
  loadi r1, 1
  dec r1
  z_flag_check_1
  loadi r1, 2
  dec r1
  flags_check_0
  ; error check
  loadi r3, test_addc
  loadi r2, 1
  comp r1, r2
  error_check

test_addc:
  loadi r6, max
  load r1, r6
  inc r1 ; overflow
  loadi r6, max
  load r1, r6
  addc r1, r0
  flags_check_1
  loadi r1, 1
  addc r1, r1
  flags_check_0
  loadi r6, max
  load r1, r6
  inc r1 ; overflow
  loadi r1, 1
  addc r1, r1
  loadi r2, 3
  ; error check
  loadi r3, test_subc
  comp r1, r2
  error_check

test_subc:
  loadi r6, max
  load r1, r6
  inc r1 ; overflow
  loadi r1, 0
  loadi r6, max
  subc r1, r2
  ;flags_check_1
  c_flag_check_1
  loadi r1, 3
  loadi r2, 1
  subc r1, r2
  flags_check_0
  loadi r6, max
  load r1, r6
  inc r1 ; overflow
  loadi r1, 3
  loadi r2, 1
  subc r1, r2
  ; error check
  loadi r3, test_comp
  loadi r2, 1
  comp r1, r2
  error_check

test_comp:
  loadi r1, 1
  comp r1, r1
  z_flag_check_1
  loadi r2, 2
  comp r1, r2
  z_flag_check_0

test_and:
  loadi r1, 0
  and r1, r1
  z_flag_check_1
  loadi r1, 1
  and r1, r1
  flags_check_0
  and r1, r0
  ; error check
  loadi r3, test_or
  comp r1, r0
  error_check

test_or:
  loadi r1, 0
  or r1, r1
  z_flag_check_1
  loadi r1, 1
  or r1, r0
  flags_check_0
  ; error check
  loadi r3, test_xor
  loadi r2, 1
  comp r1, r2
  error_check

test_xor:
  loadi r1, 0
  xor r1, r1
  z_flag_check_1
  loadi r2, 1
  xor r1, r2
  flags_check_0
  ; error check
  loadi r3, test_not
  comp r1, r2
  error_check

test_not:
  move r1, r0
  not r1
  flags_check_0
  not r1
  z_flag_check_1

test_shl:
  loadi r6, max_h
  load r1, r6
  shl r1
  flags_check_1
  loadi r1, 1
  shl r1
  flags_check_0

test_shr:
  loadi r1, 1
  shr r1
  flags_check_1
  loadi r6, max_h
  load r1, r6
  shr r1
  flags_check_0

test_shlc:
  add r0, r0 ; carry = 0
  loadi r6, max_h
  load r1, r6
  shlc r1
  flags_check_1
  loadi r6, max
  load r1, r6
  add r1, r1 ; carry = 1
  move r1, r0
  shlc r1
  flags_check_0

test_shrc:
  add r0, r0 ; carry = 0
  loadi r1, 1
  shrc r1
  flags_check_1
  loadi r6, max
  load r1, r6
  add r1, r1 ; carry = 1
  move r1, r0
  shrc r1
  flags_check_0

test_store:
  loadi r6, x
  load r1, r6
  loadi r6, store_x
  store r1, r6
  loadi r6, x
  load r1, r6
  loadi r6, store_x
  load r2, r6
  ; error check
  loadi r3, test_nop
  comp r1, r2
  error_check

test_nop:
  nop
  nop

    ;; stop simulation
done:
  loadi r4, data_start
  write_mem_dump
  sleep
```

2 Beispiel 2

2.1 Source Code

Der Sourcecode des Prol16 wurde nicht hinzugefügt, da der von der Elearning Plattform verwendet wurde.

../../src/sv/ifProl16.sv

```
interface ifProl16 #(parameter int gDataWidth = 16);

   logic [gDataWidth-1:0] mem_addr;    // unused
   logic [gDataWidth-1:0] mem_data_cpu;   // data from cpu
   logic [gDataWidth-1:0] mem_data_tb;    // data from tb
   logic mem_ce_n;
   logic mem_oe_n;
   logic mem_we_n;                     // unused
   logic illegal_inst;
   logic cpu_halt;

   modport master (
      output mem_data_tb,
      input mem_data_cpu, mem_ce_n, mem_oe_n, illegal_inst, cpu_halt
   );

endinterface
```

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

```
package pkgProl16;
   typedef bit [15:0] data_v;
endpackage
```

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

```
typedef enum bit [5:0]
   {
      Nop=6'b000000,
      Loadi=6'b000010,
      Jump=6'b001000,
      Jumpc=6'b001010,
      Jumpz=6'b001011,
      Move=6'b001100,
      And=6'b010000,
      Or=6'b010001,
      Xor=6'b010010,
      Not=6'b010011,
      Add=6'b010100,
      Addc=6'b010101,
      Sub=6'b010110,
      Subc=6'b010111,
      Comp=6'b011000,
      Inc=6'b011010,
      Dec=6'b011011,
      Shl=6'b011100,
      Shr=6'b011101,
      Shlc=6'b011110,
      Shrc=6'b011111
      //Invalid=6'b111111
   } Prol16Command;
```

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

```
`include "Prol16Command.sv"

class Prol16Opcode;
   rand int ra;
   rand int rb;
   randc Prol16Command cmd;
   rand pkgProl16::data_v data;

   constraint ra_size {ra >= 0 && ra <= 31;};
   constraint rb_size {rb >= 0 && rb <= 31;};

   function new(int ra, int rb, Prol16Command cmd, pkgProl16::data_v data);
      this.ra = ra;
      this.rb = rb;
      this.cmd = cmd;
      this.data = data;
   endfunction
endclass
```

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

```
class Prol16State #(parameter int gRegs = 32);
   pkgProl16::data_v regs[gRegs];
   pkgProl16::data_v programCounter;
   bit cFlag;
   bit zFlag;

   function new();
      for (int i = 0; i < gRegs; i++) begin
         regs[i] = '0;
      end
      programCounter = 0;
      cFlag = 0;
      zFlag = 0;
   endfunction

   task reset();
      for (int i = 0; i < gRegs; i++) begin
         regs[i] = '0;
      end
      programCounter = 0;
      cFlag = 0;
      zFlag = 0;
   endtask
endclass
```

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

```
`include "Prol16State.sv"
`include "Prol16Opcode.sv"

class Prol16Model#(parameter int gNumRegs);
   Prol16State state;

   function new(Prol16State _state);
      state = _state;
   endfunction

   task reset();
      state.reset();
   endtask

   task calcCarryFlag(int data);
      if (data > (2**$size(pkgProl16::data_v) - 1) || data < 0)
         state.cFlag = 1;
      else
         state.cFlag = 0;
   endtask

   task calcZeroFlag(pkgProl16::data_v data);
      if (data == '0)
         state.zFlag = 1;
      else
         state.zFlag = 0;
   endtask

   task execute(Prol16Opcode opcode);
      int data = 0;  // tmp data for carry calc
      bit tmpCFlag = 0; // tmp carry flag

      // inc pc here, overwrite in jumps
```

```
         state.programCounter++;

        if (opcode.ra > (gNumRegs - 1) || opcode.ra < 0 ||
            opcode.rb > (gNumRegs - 1) || opcode.rb < 0)
            $display("Illegal register, doing nop");
        else
         begin

         case (opcode.cmd)
            Nop:
            begin
            end

            Loadi:
            begin
               state.regs[opcode.ra] = opcode.data;
               state.programCounter++; // TODO: necessary?
            end

            Jump:
            begin
               state.programCounter = state.regs[opcode.ra];
            end

            Jumpc:
            begin
               if (state.cFlag == 1)
                  state.programCounter = state.regs[opcode.ra];
            end

            Jumpz:
            begin
               if (state.zFlag == 1)
                  state.programCounter = state.regs[opcode.ra];
            end

            Move:
            begin
               state.regs[opcode.ra] = state.regs[opcode.rb];
            end

            And:
            begin
               state.regs[opcode.ra] = state.regs[opcode.ra] & state.regs[
                   opcode.rb];
               state.cFlag = 0;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Or:
            begin
               state.regs[opcode.ra] = state.regs[opcode.ra] | state.regs[
                   opcode.rb];
               state.cFlag = 0;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Xor:
            begin
               state.regs[opcode.ra] = state.regs[opcode.ra] ^ state.regs[
                  opcode.rb];
               state.cFlag = 0;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Not:
            begin
               state.regs[opcode.ra] = ~(state.regs[opcode.ra]);
               state.cFlag = 0;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Add:
            begin
               data = state.regs[opcode.ra] + state.regs[opcode.rb];
               calcCarryFlag(data);
               state.regs[opcode.ra] = data;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Addc:
            begin
               data = state.regs[opcode.ra] + state.regs[opcode.rb] + state.
                  cFlag;
               calcCarryFlag(data);
               state.regs[opcode.ra] = data;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Sub:
            begin
               data = state.regs[opcode.ra] - state.regs[opcode.rb];
               calcCarryFlag(data);
               state.regs[opcode.ra] = data;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Subc:
            begin
               data = state.regs[opcode.ra] - state.regs[opcode.rb] - state.
                  cFlag;
               calcCarryFlag(data);
               state.regs[opcode.ra] = data;
               calcZeroFlag(state.regs[opcode.ra]);
            end

            Comp:
            begin
               data = state.regs[opcode.ra] - state.regs[opcode.rb];
               calcCarryFlag(data);
               calcZeroFlag(data);
            end

            Inc:
            begin
               data = state.regs[opcode.ra] + 1;
               calcCarryFlag(data);
               state.regs[opcode.ra] = data;
               calcZeroFlag(state.regs[opcode.ra]);
```

```
        end

        Dec:
        begin
           data = state.regs[opcode.ra] - 1;
           calcCarryFlag(data);
           state.regs[opcode.ra] = data;
           calcZeroFlag(state.regs[opcode.ra]);
        end

        Shl:
        begin
           if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
             state.cFlag = 1;
           else
             state.cFlag = 0;

           state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
           calcZeroFlag(state.regs[opcode.ra]);
        end

        Shr:
        begin
              if (state.regs[opcode.ra][0] == 1)
             state.cFlag = 1;
           else
             state.cFlag = 0;

              state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
           calcZeroFlag(state.regs[opcode.ra]);
        end

        Shlc:
        begin
          if (state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] == 1)
             tmpCFlag = 1;
           else
             tmpCFlag = 0;

           state.regs[opcode.ra] = state.regs[opcode.ra] << 1;
           state.regs[opcode.ra][0] = state.cFlag;
           state.cFlag = tmpCFlag;
           calcZeroFlag(state.regs[opcode.ra]);
        end

        Shrc:
        begin
          if (state.regs[opcode.ra][0] == 1)
             tmpCFlag = 1;
           else
             tmpCFlag = 0;

           state.regs[opcode.ra] = state.regs[opcode.ra] >> 1;
           state.regs[opcode.ra][$size(pkgProl16::data_v) - 1] = state.
               cFlag;
           state.cFlag = tmpCFlag;
           calcZeroFlag(state.regs[opcode.ra]);
        end

        default : $display("Wrong opcode: doing nop");
        endcase
        end
    endtask
endclass
```

../../src/sv/testProl16Rand.sv

```
`include "Prol16Model.sv"

class TestClass;
    task assertWithoutFlags(int expectedRegisterValue, int expectedPCValue,
        Prol16State state, int register, string text);
        assert (expectedRegisterValue == state.regs[register])
        else $error("Expected Register Value: %d, Actual Register Value: %d,
            Info: %s", expectedRegisterValue, state.regs[register], text);

        assert (expectedPCValue == state.programCounter)
        else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
            expectedPCValue, state.programCounter, text);
    endtask

    task assertWithFlags(int expectedRegisterValue, int expectedPCValue, int
         expectedCarryFlag, int expectedZeroFlag, Prol16State state, int
        register, string text);
        assert (expectedRegisterValue == state.regs[register])
        else $error("Expected Register Value: %d, Actual Register Value: %d,
            Info: %s", expectedRegisterValue, state.regs[register], text);

        assert (expectedPCValue == state.programCounter)
        else $error("Expected PC Value: %d, Actual PC Value: %d, Info: %s",
            expectedPCValue, state.programCounter, text);

        assert (expectedCarryFlag == state.cFlag)
        else $error("Expected Carry Flag Value: %d, Actual Carry Flag Value:
            %d, Info: %s", expectedCarryFlag, state.cFlag, text);

        assert (expectedZeroFlag == state.zFlag)
        else $error("Expected Zero Flag Value: %d, Actual Zero Flag Value: %d
            , Info: %s", expectedZeroFlag, state.zFlag, text);
    endtask

    task assertWithDuv(Prol16State state, int register, logic [15:0] cpuRegs
         [31:0], logic[15:0] cpuPc, logic cpuCFlag, logic cpuZFlag,
        Prol16Command cmd);
        assert (state.regs[register] == cpuRegs[register])
        else $error("WithDuv: Expected Register Value: %d, Actual Register
            Value: %d, Info: %s", state.regs[register], cpuRegs[register], cmd
            .name());

        assert (state.programCounter == cpuPc)
        else $error("WithDuv: Expected PC Value: %d, Actual PC Value: %d,
            Info: %s", state.programCounter, cpuPc, cmd.name());

        assert (state.cFlag == cpuCFlag)
        else $error("WithDuv: Expected Carry Flag Value: %d, Actual Carry
            Flag Value: %d, Info: %s", state.cFlag, cpuCFlag, cmd.name());

        assert (state.zFlag == cpuZFlag)
        else $error("WithDuv: Expected Zero Flag Value: %d, Actual Zero Flag
            Value: %d, Info: %s", state.zFlag, cpuZFlag, cmd.name());
    endtask
```

```
endclass

program testProl16Rand(ifProl16.master cpu, output logic rst, input logic clk);
   logic [15:0] cpuRegs [31:0];
   logic [15:0] cpuPc;
   logic cpuCFlag;
   logic cpuZFlag;
   logic [5:0] opcodeDUV;
   logic [4:0] ra;
   logic [4:0] rb;

   logic [5:0] prevOpcodeDUV;

   const int numTestCases = 100000;

   event CommandStart;
   event End;

   bit LoadiOccurred = 0;

   Prol16Opcode opcode = new(0, 0, Nop, 0);

   task trigger();
      while (!End.triggered) begin
        @(negedge cpu.mem_oe_n)
        begin
          //$display("negEdge oe");
         if (LoadiOccurred == 0)
         begin
            cpu.mem_data_tb[15:10] <= opcode.cmd;
            cpu.mem_data_tb[9:5] <= opcode.ra;
            cpu.mem_data_tb[4:0] <= opcode.rb;

         prevOpcodeDUV <= opcodeDUV;

            if (opcode.cmd == Loadi)
            begin
               LoadiOccurred = 1;
            end
         end
         else
         begin
            cpu.mem_data_tb <= opcode.data;
            LoadiOccurred = 0;
         end
        end
        @(posedge cpu.mem_oe_n)
        begin
          //$display("posEdge oe");
         -> CommandStart;
        end
      end
   endtask

   //always_ff @(negedge clk) begin
  //  previous_opcode <= opcode;
  //end

   covergroup CoverCpu @(negedge cpu.mem_oe_n);  // TODO, the right signal?
```

```
        option.per_instance = 1;

        cmd: coverpoint opcodeDUV iff (rst) {
        bins NOP   = {0};
        ignore_bins SLEEP = {1};
        bins LOADI = {2};
        ignore_bins LOAD  = {3};
        ignore_bins STORE = {4};
        bins JUMP  = {8};
        bins JUMPC = {10};
        bins JUMPZ = {11};
        bins MOVE  = {12};
        bins AND   = {16};
        bins OR    = {17};
        bins XOR   = {18};
        bins NOT   = {19};
        bins ADD   = {20};
        bins ADDC  = {21};
        bins SUB   = {22};
        bins SUBC  = {23};
        bins COMP  = {24};
        bins INC   = {26};
        bins DEC   = {27};
        bins SHL   = {28};
        bins SHR   = {29};
        bins SHLC  = {30};
        bins SHRC  = {31};
        bins Invalid = {63};
        illegal_bins other = default;
      }

        prevCmd: coverpoint prevOpcodeDUV iff (rst) {
        bins NOP   = {0};
        ignore_bins SLEEP = {1};
        bins LOADI = {2};
        ignore_bins LOAD  = {3};
        ignore_bins STORE = {4};
        bins JUMP  = {8};
        bins JUMPC = {10};
        bins JUMPZ = {11};
        bins MOVE  = {12};
        bins AND   = {16};
        bins OR    = {17};
        bins XOR   = {18};
        bins NOT   = {19};
        bins ADD   = {20};
        bins ADDC  = {21};
        bins SUB   = {22};
        bins SUBC  = {23};
        bins COMP  = {24};
        bins INC   = {26};
        bins DEC   = {27};
        bins SHL   = {28};
        bins SHR   = {29};
        bins SHLC  = {30};
        bins SHRC  = {31};
        bins Invalid = {63};
        //illegal_bins other = default;
      }

        ra: coverpoint ra;
        rb: coverpoint rb;

        crs_cmd_ra: cross cmd, ra;
        crs_cmd_rb: cross cmd, rb;

        cFlag: coverpoint cpuCFlag {
           bins set = {1};
           bins notset = {0};
        }

        zFlag: coverpoint cpuZFlag {
           bins set = {1};
           bins notset = {0};
        }

        crs_cmd_c: cross cmd, cFlag;
        crs_cmd_z: cross cmd, zFlag;

        trans_c: coverpoint cpuCFlag {
        bins from_0_to_1 = (0 => 1);
        bins from_1_to_0 = (1 => 0);
        bins from_0_to_0 = (0 => 0);
        bins from_1_to_1 = (1 => 1);
      }

        trans_z: coverpoint cpuZFlag {
        bins from_0_to_1 = (0 => 1);
        bins from_1_to_0 = (1 => 0);
        bins from_0_to_0 = (0 => 0);
        bins from_1_to_1 = (1 => 1);
      }

        prevCmdChangeFlags: coverpoint prevOpcodeDUV {
        bins doesnt_change_flags = { 0, 1, 2, 3, 4, 8, 10, 11, 12};
        bins c_flag_0_and_z_flag_x = { 16, 17, 18, 19 };
        bins c_flag_x_and_z_flag_x = { 20, 21, 22, 23, 24, 26, 27, 28, 29,
            30, 31 };
      }

        crs_cmd_c_change: cross prevCmdChangeFlags, trans_c {
        illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
            (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_0));

      }

        crs_cmd_z_change: cross prevCmdChangeFlags, trans_z {
        illegal_bins ill = binsof(prevCmdChangeFlags.doesnt_change_flags) &&
            (binsof(trans_z.from_0_to_1) || binsof(trans_z.from_1_to_0));
      }

      crs_cmd_c_is_0: cross prevCmdChangeFlags, trans_c {
        illegal_bins ill = binsof(prevCmdChangeFlags.c_flag_0_and_z_flag_x)
            && (binsof(trans_c.from_0_to_1) || binsof(trans_c.from_1_to_1));
      }

     endgroup

     initial begin : stimuli
        Prol16State state = new();
```

```
        Prol16Model#(32) model = new(state);

        TestClass testClass = new();

        CoverCpu coverCpu = new;

        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(0)"
            , "/top/TheTest/cpuRegs(0)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(1)"
            , "/top/TheTest/cpuRegs(1)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(2)"
            , "/top/TheTest/cpuRegs(2)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(3)"
            , "/top/TheTest/cpuRegs(3)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(4)"
            , "/top/TheTest/cpuRegs(4)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(5)"
            , "/top/TheTest/cpuRegs(5)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(6)"
            , "/top/TheTest/cpuRegs(6)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(7)"
            , "/top/TheTest/cpuRegs(7)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(8)"
            , "/top/TheTest/cpuRegs(8)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(9)"
            , "/top/TheTest/cpuRegs(9)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(10)
            ", "/top/TheTest/cpuRegs(10)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(11)
            ", "/top/TheTest/cpuRegs(11)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(12)
            ", "/top/TheTest/cpuRegs(12)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(13)
            ", "/top/TheTest/cpuRegs(13)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(14)
            ", "/top/TheTest/cpuRegs(14)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(15)
            ", "/top/TheTest/cpuRegs(15)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(16)
            ", "/top/TheTest/cpuRegs(16)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(17)
            ", "/top/TheTest/cpuRegs(17)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(18)
            ", "/top/TheTest/cpuRegs(18)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(19)
            ", "/top/TheTest/cpuRegs(19)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(20)
            ", "/top/TheTest/cpuRegs(20)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(21)
            ", "/top/TheTest/cpuRegs(21)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(22)
            ", "/top/TheTest/cpuRegs(22)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(23)
            ", "/top/TheTest/cpuRegs(23)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(24)
            ", "/top/TheTest/cpuRegs(24)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(25)
            ", "/top/TheTest/cpuRegs(25)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(26)
            ", "/top/TheTest/cpuRegs(26)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(27)", "/top/TheTest/cpuRegs(27)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(28)", "/top/TheTest/cpuRegs(28)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(29)", "/top/TheTest/cpuRegs(29)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(30)", "/top/TheTest/cpuRegs(30)");
        $init_signal_spy("/top/TheCpu/datapath_inst/thereg_file/registers(31)", "/top/TheTest/cpuRegs(31)");

        $init_signal_spy("/top/TheCpu/datapath_inst/RegPC", "/top/TheTest/cpuPc");
        $init_signal_spy("/top/TheCpu/control_inst/Carry", "/top/TheTest/cpuCFlag");
        $init_signal_spy("/top/TheCpu/control_inst/Zero", "/top/TheTest/cpuZFlag");
        $init_signal_spy("/top/TheCpu/datapath_inst/RegOpcode", "/top/TheTest/opcodeDUV");
        $init_signal_spy("/top/TheCpu/datapath_inst/RegAIdx", "/top/TheTest/ra");
        $init_signal_spy("/top/TheCpu/datapath_inst/RegBIdx", "/top/TheTest/rb");

        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(0)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(1)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(2)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(3)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(4)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(5)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(6)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(7)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(8)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(9)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(10)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(11)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(12)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(13)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(14)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(15)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(16)", "16#0000", 0, 1);
        $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(17)", "16#0000", 0, 1);
```

```
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(18)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(19)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(20)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(21)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(22)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(23)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(24)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(25)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(26)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(27)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(28)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(29)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(30)", "16#0000", 0, 1);
    $signal_force("/top/TheCpu/datapath_inst/thereg_file/registers(31)", "16#0000", 0, 1);


    // generate reset ------------------------------------------------------
    rst = 1;
    #10 rst = 0;
    #20 rst = 1;

    //Reset
    model.reset();
    //testClass.assertWithoutFlags(0, 0, model.state, 12, "Reset test");

    fork
      trigger();
    join_none

  //Nop
    @(CommandStart);
    testClass.assertWithDuv(model.state, 12, cpuRegs, cpuPc, cpuCFlag, cpuZFlag, opcode.cmd);


    for (int i = 0; i < numTestCases; i++) begin
      if (opcode.cmd == Loadi)
      begin
         @(CommandStart);
      end
      model.execute(opcode);
      opcode.randomize();
      //$display("Command %b", opcode.cmd);
      //$display("CommandDUV %b", opcodeDUV);
       @(CommandStart);
```

```
         testClass.assertWithDuv(model.state, opcode.ra, cpuRegs, cpuPc,
            cpuCFlag, cpuZFlag, opcode.cmd);
      end

      -> End;
      $stop;
   end : stimuli
endprogram
```

../../src/sv/top.sv

```
`include "testProl16Rand.sv"

module top;
   logic clk = 0, rst;
   always #10 clk = ~clk;

   ifProl16 ifCpu();
   cpu TheCpu(clk, rst, ifCpu.mem_addr, ifCpu.mem_data_cpu, ifCpu.
      mem_data_tb, ifCpu.mem_ce_n, ifCpu.mem_oe_n,
            ifCpu.mem_we_n, ifCpu.illegal_inst, ifCpu.cpu_halt);
   testProl16Rand TheTest(ifCpu.master, rst, clk);
endmodule
```

../../sim/CompileSimRandCov.do

```
vlib work

vlog ../src/sv/ifProl16.sv
vlog ../src/sv/pkgProl16.sv
vlog ../src/sv/Prol16Command.sv
vlog ../src/sv/Prol16Opcode.sv +incdir+../src/sv
vlog ../src/sv/Prol16State.sv +incdir+../src/sv
vlog ../src/sv/Prol16Model.sv +incdir+../src/sv
vlog ../src/sv/testProl16Rand.sv +incdir+../src/sv
vlog ../src/sv/top.sv +incdir+../src/sv

vcom -87 ../src/vhdl/prol16_pack.vhd
vcom -87 ../src/vhdl/reg_file.vhd
vcom -87 ../src/vhdl/datapath.vhd
vcom -87 ../src/vhdl/control.vhd
vcom -87 ../src/vhdl/alu.vhd
vcom -93 ../src/vhdl/memory.vhd
vcom -87 ../src/vhdl/cpu.vhd
#vcom -87 ../src/vhdl/cpu_tb.vhd

vsim -novopt top
run -all
```