NC State University

Department of Electrical and Computer Engineering

ECE 745 ASIC VERIFICATION

Fall 2015

LC3 MICROCONTROLLER VERIFICATION COVERAGE REPORT

Group 23

Introduction

The objective of the project was to completely verify the LC3 microcontroller using a reusable and fully layered test-bench for the verification.
A standard and efficient environment structure was created to finish the verification task. In the structure different System Verilog classes/modules/programs are used to create the transaction, generator, driver, environment, scoreboard and so on. The transactions are communicated between the classes using the SV Mailboxes with handshakes controlling synchronization. Golden references are created in

Running Procedures

1. Open the Verification file and add all the required .vp and .v files.
2. Open the code in terminal and perform add modelsim10.3b and setenv MODELSIM modelsim.ini.
3. Then do a vlog of the .v and .vp files using the following commands.
   vlog *.v
   vlog *.vp

4. Since the modelsim.ini we are using is of the version mti_lib, use the following vlib command
   vlib mti_lib

5. Then to compile the .sv files use the following command
   vlog -sv Testbench.sv

6. Then for vsim, do the following
   vsim -novopt top_testbench &

7. Once Modelsim opens, just use the compile.do command and this will generate the waveforms and run the simulation.
8. Let the simulation run for at least 2,000,000 ns. After this time the coverage will saturate.

Cover Groups

4 Cover Groups are created, MEM_OPR_cg, OPR_SEQ_cg, ALU_OPR_cg and CTRL_OPR_cg. In each of them different cover points are created. The definitions all all following the rules provided in Verification_Plan.doc by Teaching Assistant, however, some changes or special attention have been made to some of them shown below.

1. Cover Group MEM_OPR_cg

Cover Point	**Description**
Cov_mem_opcode	bins LD_b = {LD }; bins LDR_b = {LDR}; bins LDI_b = {LDI}; bins LEA_b = {LEA}; bins ST_b = {ST };

	bins STR_b = {STR}; bins STI_b = {STI};
Cov_BaseR_LDR	bins LDR_BaseR[] = {[3'b000:3'b111]};
Cov_BaseR_STR	bins STR_BaseR[] = {[3'b000:3'b111]};
Cov_SR_ST	bins ST_SR[] = {[3'b000:3'b111]};
Cov_SR_STI	bins STI_SR[] = {[3'b000:3'b111]};
Cov_SR_STR	bins STR_SR[] = {[3'b000:3'b111]};
Cov_DR_LD	bins LD_DR[] = {[3'b000:3'b111]};
Cov_DR_LDR	bins LDR_DR[] = {[3'b000:3'b111]};
Cov_DR_LDI	bins LDI_DR[] = {[3'b000:3'b111]};
Cov_DR_LEA	bins LEA_DR[] = {[3'b000:3'b111]};
Cov_PCoffset9	option.auto_bin_max = 8 ;
Cov_PCoffset6_LDR & STR	option.auto_bin_max = 8 ;
Cov_PCoffset9_c	bins PCoffset9_c_high= {9'b111111111}; bins PCoffset9_c_low= {9'b000000000};
Cov_PCoffset6_c	bins PCoffset6_c_high= {6'b111111}; bins PCoffset6_c_low= {6'b000000};
Xc_BaseR_DR_offset6 //only covers PCoffset6_LDR	ignore_bins others = binsof(Cov_mem_opcode) intersect {LDR, LD, LDI, STI, ST, LEA}
Xc_BaseR_SR_offset6 //only covers PCoffset6_STR	ignore_bins others = binsof(Cov_mem_opcode) intersect {STR, LD, LDI, STI, ST, LEA};

2. Cover Group ALU_OPR_cg

Cover Point	**Description**
Cov_alu_opcode	bins ADD_b = {ADD}; bins AND_b = {AND}; bins NOT_b = {NOT};
Cov_alu_opcode_AND_ADD //Instead of directly set cover point for the entire alu_opcode, we split them into two groups to	bins AND_p = {AND}; bins ADD_p = {ADD};

serve Xc_opcode_imm_en, Xc_opcode_dr_sr1_imm5, Xc_opcode_dr_sr1_sr2, Xc_Cov_opr_zero_ALL1 and Xc_Cov_opr_pos_neg.	
Cov_alu_opcode_NOT	bins NOT_p = {NOT};
Cov_imm_en	bins HIGH = {1'b1}; bins LOW = {1'b0}
Cov_SR1	bins ALU_SR1[] = {[0:7]};
Cov_SR2	bins ALU_SR2[] = {[0:7]};
Cov_DR	bins ALU_DR[] = {[0:7]};
Cov_imm5	bins IMM[] = {[5'b00000:5'b11111]};
Xc_opcode_imm_en //only covers Cov_alu_opcode_AND_ADD	cross Cov_alu_opcode_AND_ADD,Cov_imm_en;
Xc_opcode_dr_sr1_imm5 //only covers Cov_alu_opcode_AND_ADD	cross Cov_alu_opcode_AND_ADD,Cov_SR1,Cov_DR,Cov_imm5 iff(ev.de_gr.Instr_dout[5] == 1'b1);
Xc_opcode_dr_sr1_sr2 //only covers Cov_alu_opcode_AND_ADD	cross Cov_alu_opcode_AND_ADD,Cov_SR1,Cov_SR2,Cov_DR iff(ev.de_gr.Instr_dout[5] == 1'b0);
Cov_aluin1	option.auto_bin_max = 8 ;
Cov_aluin1_corner	bins aluin1_corner_high = {16'h0000} ; bins aluin1_corner_low = {16'hFFFF} ;
Cov_aluin2	option.auto_bin_max = 8 ;
Cov_aluin2_corner	bins aluin2_corner_high = {16'h0000} ; bins aluin2_corner_low = {16'hFFFF} ;
Xc_opcode_aluin1	cross Cov_alu_opcode,Cov_aluin1_corner;
Xc_opcode_aluin2	cross Cov_alu_opcode,Cov_aluin2_corner;
Xc_Cov_opr_zero_ALL1 //only covers Cov_alu_opcode_AND_ADD	cross Cov_alu_opcode_AND_ADD,Cov_aluin1_corner,Cov_aluin2_corner;
Cov_aluin1_alt01 //0 coverage	bins aluin1_alt01 = {16'b0101010101010101} ; bins aluin1_alt10 = {16'b1010101010101010} ; bins others = default;

Cov_aluin2_alt01 //0 coverage	bins aluin2_alt01 = {16'b0101010101010101} ; bins aluin2_alt10 = {16'b1010101010101010} ; bins others = default;
Xc_Cov_opr_alt01_alt10 //0 coverage	cross Cov_alu_opcode,Cov_aluin1_alt01,Cov_aluin2_alt01 iff(ev.de_gr.Instr_dout[15:12] != NOT);
Cov_aluin1_pos_neg	bins aluin1_pos = {1'b0} ; bins aluin1_neg = {1'b1} ;
Cov_aluin2_pos_neg	bins aluin2_pos = {1'b0} ; bins aluin2_neg = {1'b1} ;
Xc_Cov_opr_pos_neg //only covers Cov_alu_opcode_AND_ADD	cross Cov_alu_opcode_AND_ADD,Cov_aluin1_pos_neg,Cov_aluin2_pos_neg;

3. Cover Group CTRL_OPR_cg

Cover Point	**Description**
Cov_ctrl_opcode	bins BR_b = {BR}; bins JMP_b = {JMP};
Cov_BaseR	bins BaseR_JMP[] = {[0:7]};
Cov_NZP	bins NZP_JMP = {7};
Cov_PSR	bins psr_JMP_1 = {1}; bins psr_JMP_2 = {2}; bins psr_JMP_4 = {4};
Cov_PCoffset9	option.auto_bin_max = 8 ;
Cov_PCoffset9_c	bins pcoffset9_corner_low = {9'h000} ; bins pcoffset9_corner_high = {9'h1FF} ;
Xc_NZP_PSR	cross Cov_NZP,Cov_PSR;

4. Cover Group OPR_SEQ_cg

Cover Point	**Description**
Cov_opcode_order	bins ALU_Memory = (AND, ADD, NOT =>

	LD, LDR, LDI, LEA, ST, STR, STI);
	bins Memory_Control = (LD, LDR, LDI, LEA, ST, STR, STI => AND, ADD, NOT);
	bins Memory_ALU = (LD, LDR, LDI, LEA, ST, STR, STI => AND, ADD, NOT);

Cov_mem_opcode_temp	bins LD_b = {LD }; bins LDR_b = {LDR}; bins LDI_b = {LDI}; bins LEA_b = {LEA}; bins ST_b = {ST }; bins STR_b = {STR}; bins STI_b = {STI};
Cov_alu_opcode_temp	bins ADD_b = {ADD}; bins AND_b = {AND}; bins NOT_b = {NOT};
Cov_ctrl_opcode_temp	bins BR_b = {BR}; bins JMP_b = {JMP};

Cover Properties

Assertions were also coded in the probe interface file to verify the different scenarios and sequences of the DUT (mainly the controller). They are being covered to make sure these assertions have been exercised by the test cases. In our project we added 8 kinds of properties in the test bench to assert some important properties.

1. Reset Behaviour:

```
property reset_if; reset |-> ##1 (pc == 16'h3000);

property reset_dc; reset |-> ##1
(!((Mem_Control)||(W_Control)||(E_Control)||(IR)||(npc_out)));

property reset_ex; reset |-> ##1
(!((Mem_Control_out)||(W_Control_out)||(dr)||(NZP)||(IR_Exec)||(aluout)||(pcout)||(M_Data)))
;

property reset_wb; reset |-> ##1 (!psr);

property reset_ctrl; reset |-> ##1
(!((enable_decode)||(enable_execute)||(enable_writeback)))&&(!(enable_updatePC)&&(enabl
e_fetch)&&(mem_state==2'b11));
```

2. Br_taken:

```
property CTRL_br_taken_jmp;
|(NZP & psr) |-> br_taken;
```

3. Enable_Fetch, Enable_decode, Enable_execute

```
property CTRL_enable_fetch_low;
```

```
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 || IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011 || IMem_dout[15:12] == 4'b0000 ||
IMem_dout[15:12] == 4'b1100) |=> !enable_fetch;

property CTRL_enable_fetch_rise1;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 ) |=> ##1 enable_fetch;

property CTRL_enable_fetch_rise2;
 (IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011) |=> ##2 enable_fetch;

property CTRL_enable_fetch_rise3;
 (IMem_dout[15:12] == 4'b0000) || (IMem_dout[15:12] == 4'b1100) |=> ##3
enable_fetch;

property CTRL_enable_decode_fall1;
(IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 || IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011 ) |=> !enable_decode;

property CTRL_enable_decode_fall2;
 (IMem_dout[15:12] == 4'b0000 || IMem_dout[15:12] == 4'b1100) |=> ##1
!enable_decode;

property CTRL_enable_decode_rise1;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 ) |=> ##1 enable_decode;

property CTRL_enable_decode_rise2;
 (IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011) |=> ##2 enable_decode;

property CTRL_enable_decode_rise3;
 (IMem_dout[15:12] == 4'b0000) || (IMem_dout[15:12] == 4'b1100) |=> ##4
enable_decode;

property CTRL_enable_execute_fall1;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 || IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011 ) |=> !enable_execute;

property CTRL_enable_execute_fall2;
(IMem_dout[15:12] == 4'b0000 || IMem_dout[15:12] == 4'b1100) |=> ##2
!enable_execute;

property CTRL_enable_execute_rise1;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 ) |=> ##1 enable_execute;

property CTRL_enable_execute_rise2;
 (IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011) |=> ##2 enable_execute;

property CTRL_enable_execute_rise3;
 (IMem_dout[15:12] == 4'b0000) || (IMem_dout[15:12] == 4'b1100) |=> ##5
```

```
enable_execute;

property CTRL_enable_writeback_fall1;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b0010 || IR[15:12] ==
4'b0110 || IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011 ) |=> !enable_writeback;

property CTRL_enable_writeback_fall2;
 (IMem_dout[15:12] == 4'b0000 || IMem_dout[15:12] == 4'b1100) |=> ##2
!enable_writeback;

property CTRL_enable_writeback_rise1;
(IR[15:12] == 4'b0010 || IR[15:12] == 4'b0110 ) |=> ##1 enable_writeback;

property CTRL_enable_writeback_rise2;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 || IR[15:12] == 4'b1010) |=> ##2
enable_writeback;

property CTRL_enable_writeback_rise3;
 (IR[15:12] == 4'b1011) |=> ##3 enable_writeback;

property CTRL_enable_writeback_rise4;
 (IMem_dout[15:12] == 4'b0000) || (IMem_dout[15:12] == 4'b1100) |=> ##6
enable_writeback;
```

4. Bypass_alu:

```
property CTRL_bypass_alu_1_AA;
 ((IR[15:12] == 4'b0001) || (IR[15:12] == 4'b0101) || (IR[15:12] == 4'b1001)) &&
((IR_Exec[15:12] == 4'b0001) || (IR_Exec[15:12] == 4'b0101) || (IR_Exec[15:12] ==
4'b1001)
|| (IR_Exec[15:12] == 4'b1110)) && (IR_Exec[11:9] == IR[8:6]) |-> bypass_alu_1 == 1'b1;

property CTRL_bypass_alu_2_AA;
 ((IR[15:12] == 4'b0001) || (IR[15:12] == 4'b0101) || (IR[15:12] == 4'b1001)) &&
((IR_Exec[15:12] == 4'b0001) || (IR_Exec[15:12] == 4'b0101) || (IR_Exec[15:12] ==
4'b1001)
|| (IR_Exec[15:12] == 4'b1110)) && ((IR_Exec[11:9] == IR[2:0]) && (IR[5] == 1'b0)) |->
bypass_alu_2 == 1'b1;

property CTRL_bypass_alu_1_AL;
 ((IR_Exec[15:12] == 4'b0001) || (IR_Exec[15:12] == 4'b0101) || (IR_Exec[15:12] ==
4'b1001)) && (IR[15:12] == 4'b0110) && (IR_Exec[11:9] == IR[8:6]) |-> bypass_alu_1 ==
1'b1;

property CTRL_bypass_alu_1_AS;
 ((IR_Exec[15:12] == 4'b0001) || (IR_Exec[15:12] == 4'b0101) || (IR_Exec[15:12] ==
4'b1001)) && (IR[15:12] == 4'b0111) && (IR_Exec[11:9] == IR[8:6]) |-> bypass_alu_1 ==
1'b1;
```

5. Bypass_mem:

```
property CTRL_bypass_mem_1_LA;
 ((IR_Exec[15:12] == 4'b0010) || (IR_Exec[15:12] == 4'b0110) || (IR_Exec[15:12] ==
```

```
4'b1010)) && ((IR[15:12] == 4'b0001) || (IR[15:12] == 4'b0101) || (IR[15:12] == 4'b1001))
&&
(IR_Exec[11:9] == IR[8:6]) |-> bypass_mem_1 == 1'b1;

property CTRL_bypass_mem_2_LA;
 ((IR_Exec[15:12] == 4'b0010) || (IR_Exec[15:12] == 4'b0110) || (IR_Exec[15:12] ==
4'b1010)) && ((IR[15:12] == 4'b0001) || (IR[15:12] == 4'b0101) || (IR[15:12] == 4'b1001))
&&
((IR_Exec[11:9] == IR[2:0]) && (IR[5] == 1'b0)) |-> bypass_mem_2 == 1'b1;
```

6. Memory State Machine Coverage:

```
property CTRL_mem_state_3_1;
 (mem_state == 2'b11) && ((IR[15:12] == 4'b1010) || (IR[15:12] == 4'b1010))
|=> mem_state == 2'b01;

property CTRL_mem_state_3_0;
 (mem_state == 2'b11) && (IR[15:12] == 4'b1010) |=> mem_state == 2'b01 ##1
mem_state == 2'b00;

property CTRL_mem_state_3_2;
 (mem_state == 2'b11) && (IR[15:12] == 4'b1011) |=> mem_state == 2'b01 ##1
mem_state == 2'b10;

property CTRL_mem_state_2_3;
 (mem_state == 2'b10) && (complete_data == 1) |=> mem_state == 2'b11;
property CTRL_mem_state_0_3;
 (mem_state == 2'b00) && (complete_data == 1) |=> mem_state == 2'b11;

property CTRL_mem_state_1_2;
 (mem_state == 2'b01) && (IR_Exec[15:12] == 4'b1011) |=> mem_state == 2'b10;

property CTRL_mem_state_1_0;
 (mem_state == 2'b01) && (IR_Exec[15:12] == 4'b1010) |=> mem_state == 2'b00;
```

7. Memory State Machine flow:

```
property CTRL_enable_mem_state_LDI;
 (IR[15:12] == 4'b1010) |=> mem_state == 2'b01 ##1 mem_state == 2'b00 ##1
mem_state == 2'b11;

property CTRL_enable_mem_state_STI;
 (IR[15:12] == 4'b1011) |=> mem_state == 2'b01 ##1 mem_state == 2'b10 ##1
mem_state == 2'b11;

property CTRL_enable_mem_state_STI_LDI;
 (IR[15:12] == 4'b1010 || IR[15:12] == 4'b1011 ) |=> mem_state == 2'b01 ##2
mem_state == 2'b11;

property CTRL_enable_mem_state_ST_STR;
 (IR[15:12] == 4'b0011 || IR[15:12] == 4'b0111 ) |=> mem_state == 2'b10 ##1
mem_state == 2'b11;
```

```
property CTRL_enable_mem_state_LD_LDR;
 (IR[15:12] == 4'b0010 || IR[15:12] == 4'b0110 ) |=> mem_state == 2'b00 ##1
mem_state == 2'b11;
```

8. Writeback Enable after STORE and LOAD

```
property CTRL_enable_wb_ST;
 (IMem_dout[15:12] == 4'b0111 || IMem_dout[15:12] == 4'b0011) |=> ##2
!enable_writeback ##2 enable_writeback;
```

Simulation Results

The test bench needs to run many enough simulations to reach as high coverage as possible. In our own tests, after running enough time (larger than 2,000,000 ns) the coverage will saturate into 98.95%.

Recursive Hierarchical Coverage Details:

Total Coverage:					99.75%	98.95%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	6421	6405	16	1	99.75%	**96.87%**
Directives	45	45	0	1	100.00%	**100.00%**
Assertions	69	69	0	1	100.00%	**100.00%**

It's not reaching 100% because and only because alt01 and alt02 under ALU_OPG_cg cover group are not covered (and surely corresponding cross cover points including them). The screenshots of the coverage report will show this 0 coverage below. We can, however, add directive test cases to make it reach 100%.

1. Cover Group MEM_OPR_cg (reaches 100%)

Covergroup type:MEM_OPR_cg					100.00%	100.00%	100.00%
Coverpoints / Bins	**At Least**	**Total Bins**	**Covered Bins (Hits)**	**% Hit**	**Goal**	**Coverage**	**% of Goal**
Coverpoint: Cov_mem_opcode		7	7	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_BaseR_LDR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_BaseR_STR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_SR_ST		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_SR_STI		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_SR_STR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_DR_LD		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_DR_LDR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_DR_LDI		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_DR_LEA		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset9		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset6		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset9_c		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset6_c		2	2	100.00%	100.00%	100.00%	100.00%
Cross: Xc_BaseR_DR_offset6 <Cov_PCoffset6, Cov_SR_STR, Cov_BaseR_STR, Cov_mem_opcode>		512	512	100.00%	100.00%	100.00%	100.00%
Cross: Xc_BaseR_SR_offset6 <Cov_PCoffset6, Cov_DR_LDR, Cov_BaseR_LDR, Cov_mem_opcode>		512	512	100.00%	100.00%	100.00%	100.00%

2. Cover Group ALU_cg (not reaching 100%)

Covergroup type:ALU_OPR_cg				100.00%	87.49%	87.49%
Coverpoints / Bins	At Least	Total Bins	Covered Bins (Hits)	% Hit	Goal	Coverage	% of Goal
Coverpoint: Cov_alu_opcode		3	3	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_alu_opcode_AND_ADD		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_alu_opcode_NOT		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_imm_en		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_SR1		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_SR2		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_DR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_imm5		32	32	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin1		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin1_corner		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin2		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin2_corner		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin1_alt01		2	0	0.00%	100.00%	0.00%	0.00%
Coverpoint: Cov_aluin2_alt01		2	0	0.00%	100.00%	0.00%	0.00%
Coverpoint: Cov_aluin1_pos_neg		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_aluin2_pos_neg		2	2	100.00%	100.00%	100.00%	100.00%
Cross: Xc_opcode_imm_en <Cov_alu_opcode_AND_ADD, Cov_imm_en>		4	4	100.00%	100.00%	100.00%	100.00%
Cross: Xc_opcode_dr_sr1_imm5 <Cov_alu_opcode_AND_ADD, Cov_SR1, Cov_DR, Cov_imm5>		4096	4089	99.82%	100.00%	99.82%	99.82%
Cross: Xc_opcode_dr_sr1_sr2 <Cov_alu_opcode_AND_ADD, Cov_SR1, Cov_SR2, Cov_DR>		1024	1024	100.00%	100.00%	100.00%	100.00%
Cross: Xc_opcode_aluin1 <Cov_alu_opcode, Cov_aluin1_corner>		6	6	100.00%	100.00%	100.00%	100.00%
Cross: Xc_opcode_aluin2 <Cov_alu_opcode, Cov_aluin2_corner>		6	6	100.00%	100.00%	100.00%	100.00%
Cross: Xc_Cov_opr_zero_ALL1 <Cov_alu_opcode_AND_ADD, Cov_aluin1_corner, Cov_aluin2_corner>		8	8	100.00%	100.00%	100.00%	100.00%
Cross: Xc_Cov_opr_alt01_alt10 <Cov_alu_opcode, Cov_aluin1_alt01, Cov_aluin2_alt01>		12	0	0.00%	100.00%	0.00%	0.00%
Cross: Xc_Cov_opr_pos_neg <Cov_alu_opcode_AND_ADD, Cov_aluin1_pos_neg, Cov_aluin2_pos_neg>		8	8	100.00%	100.00%	100.00%	100.00%

3. Cover Group CTRL_OPR_cg (reaches 100%)

Covergroup type:CTRL_OPR_cg				100.00%	100.00%	100.00%
Coverpoints / Bins	At Least	Total Bins	Covered Bins (Hits)	% Hit	Goal	Coverage	% of Goal
Coverpoint: Cov_ctrl_opcode		2	2	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_BaseR		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_NZP		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PSR		3	3	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset9		8	8	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_PCoffset9_c		2	2	100.00%	100.00%	100.00%	100.00%
Cross: Xc_NZP_PSR <Cov_NZP, Cov_PSR>		3	3	100.00%	100.00%	100.00%	100.00%

4. Cover Group OPR_SEQ_cg (reaches 100%)

Covergroup type:OPR_SEQ_cg				100.00%	100.00%	100.00%
Coverpoints / Bins	At Least	Total Bins	Covered Bins (Hits)	% Hit	Goal	Coverage	% of Goal
Coverpoint: Cov_opcode_order		3	3	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_mem_opcode_temp		7	7	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_alu_opcode_temp		3	3	100.00%	100.00%	100.00%	100.00%
Coverpoint: Cov_ctrl_opcode_temp		2	2	100.00%	100.00%	100.00%	100.00%

The detailed coverage reports are in the zip file, named report_default.txt and report_details.sv.