**LC3 Microcontroller**

**Verification Plan and Coverage Report**

Samarth Asthana, Nikhil Khatu, Sonam Chakravarti

**1. Introduction**

Our Device Under Test is the Von-Neumann based 16-bit LC3 Microcontroller that takes instruction values, and then processes these values through Fetch, Decode, Writeback, Memory\_Access, and Execute stages. This document basically explains the test bench strategy we have followed to verify the DUT, with detailed analysis of coverage achieved and how it was improved using directed test cases. The verification process was divided into three main parts:

1) Coverage Groups

2) Coverage Properties

3) Directed Test Cases

Since the inputs were generated randomly and there are at least 10 ALU operations before and after any control or memory instructions, it was necessary to test few conditions to ensure complete functional coverage. These mainly included the data hazards caused in a pipelined architecture (data dependencies), branch taken, and not taken for control instructions. The test bench was provided with variable seeds in incremental order to reach the achieved coverage. Most of the testing was done using constrained random testing and directed test cases were written only to reach corner cases and few scenarios which otherwise would not have been covered. These are discussed in detail in next few sections.

Also while designing the generator, it was kept in mind that not all instructions are valid for current DUT and scoreboard and they were avoided in generation phase itself. Each module has its own reset behavior well defined in the scoreboard and it is validated at the beginning of the simulation. While the former is tested using cover groups, latter is done using cover properties. The directed test cases run at the beginning of the simulation (around 50 cases covering up to 10 scenarios) while the remaining is constrained random testing.

**2. Cover Groups Used**

The cover groups were divided mainly into 5 parts:

**1. Arithmetic Coverage : covergroup ALU\_OPR\_cg**

This cover group covers all the cover points for ALU instructions (i.e. opcode, destination reg, source 1 and 2, immediate mode, reg mode etc.). Few of the cover points from given sample were combined to one cover point for different bins. A separate bin was created for every corner case possible. The cover points for the group with details are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. used | Coverage |
| 1. | ALU\_OP | pkt\_sent.op | Bins are created for “ADD”, “AND” and “NOT” Instructions |
| 2. | bit5 | pkt\_sent.bit5 | Bins for register mode and immediate mode |
| 3. | sr1 | pkt\_sent.sr1 | Bins for all 8 registers for SR1 |
| 4. | im\_sr2 | coverpoint pkt\_sent.im\_sr2 | Bins for all 8 registers for SR2 |
| 5. | dr | pkt\_sent.dr | Bins for all 8 registers for DR |
| 6. | pcof | pkt\_sent.pcof | PCOffset 9 –Bins for All zeros, All ones, Remaining 8 bins |
| 7. | aluin1 | ex\_alu\_in1 | Bins for All zeros, All ones, Alternating Zero’s and One’s Alternating One’s and Zero’s, positive values, negative values |
| 8. | aluin2 | ex\_alu\_in2 | Bins for All zeros, All ones, Alternating Zero’s and One’s Alternating One’s and Zero’s, positive values, negative values |
| 9. | ALU\_IMM5 | cross ALU\_OP,bit5 | Bins for instructions with immediate and register mode. Not instruction ignored |
| 10. | IMM\_MODE | cross ALU\_OP,dr,sr1,bit5,im\_sr2 | Cross to check combinations of immediate mode instructions |
| 11. | REG\_MODE | cross ALU\_OP,sr1,dr,bit5,im\_sr2 | Cross to check combinations of register mode instructions |
| 12. | CORNER\_ALUIN1 | cross ALU\_OP,aluin1 | Corner cases to check values of alunin1 (all ones and all zeros) |
| 13. | CORNER\_ALUIN2 | cross ALU\_OP,aluin2 | Corner cases to check values of alunin2 (all ones and all zeros) |
| 14. | CORNER\_ALUIN\_1\_2 | cross ALU\_OP,aluin1,aluin2 | Corner cases to check values of aluin1 and alunin2 (all ones and all zeros) |

**2. Memory Coverage : covergroup MEM\_OPR\_cg**

The memory instructions were covered in cover properties to make sure that all possible combinations of valid memory instructions are exercised.

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. used | Coverage |
| 1. | MEM\_OP | pkt\_sent.op | Bins of all memory operations |
| 2. | BaseR | pkt\_sent.im\_sr2[2:0] | Bins for all possible values of base R including corner cases |
| 3. | PC9 | pkt\_sent.pcof | All possible values including corner cases and remaining values are divided into 8 bins |
| 4. | SR | pkt\_sent.dr | Bins for all possible values including corner cases |
| 5. | PC6 | {pkt\_sent.bit5,pkt\_sent.im\_sr2} | All possible values including corner cases and remaining values are divided into 8 bins |
| 6. | LD\_ST | cross MEM\_OP,PC9 | Cross coverage for LD and ST instructions with pc offset 9 |
| 7. | SR\_ST | cross SR,MEM\_OP | Cross coverage for store instructions with SR |
| 8. | DR\_LD | cross SR,MEM\_OP | Cross coverage for load instructions with DR |
| 9. | SR\_BaseR\_LDR | cross MEM\_OP,SR,BaseR,PC6 | Cross coverage for LDR instructions with source base R and PC Offset 6 |
| 10. | SR\_BaseR\_STR | cross MEM\_OP,SR,BaseR,PC6 | Cross coverage for STR instructions with source base R and PC Offset 6 |

**3. Control Coverage: covergroup CTRL\_OPR\_cg**

The control instructions JMP and BR were included in this covergroup to exercise the valid instruction combinations and avoid invalid ones

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. Used | Coverage |
| 1. | Cov\_ctrl\_opcode | pkt\_sent.op | Bins of all Control instruction opcodes |
| 2. | Cov\_BaseR | pkt\_sent.sr1 | Bin for JMP instruction Base R |
| 3. | Cov\_NZP | control\_nzp | Bins for all possible values of NZP |
| 4. | Cov\_PSR | control\_psr | Bins for all possible values of PSR |
| 5. | Cov\_PCoffset9 | pkt\_sent.pcof | Bins for all possible values of PC Offset 9 for branch instructions |
| 6. | Cov\_PCoffset9\_c | pkt\_sent.pcof | Bins for corner cases of PC Offset 9 for branch instructions |
| 7. | Xc\_NZP\_PSR | cross Cov\_NZP,Cov\_PSR | Combinations of NZP and PSR covered which decide whether that branch is taken or not |

**4. Sequence Coverage: covergroup OPR\_SEQ\_cg**

The sequence coverage group covered the transitions of different types of instructions to make sure all possible combinations are exercised like ALU=>CNTRL, ALU=>MEM,CNTRL=>ALU,MEM=>ALU etc.

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. used | Coverage |
| 1. | SEQ\_OP | pkt\_sent.op | Bins for all possible transitions are carefully chosen and created. The above snippet shows the transitions considered which are ALU=>CNTRL, ALU=>MEM,CNTR=>ALU,MEM=>ALU etc. |

**5. Internal Signal Coverage:**

1. covergroup DEC\_INT\_SIG\_cg

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. Used | Coverage |
| 1. | E\_CNTRL | pkt2cmp.decode\_E\_control | Bins for all possible values including corner cases |
| 2. | W\_CNTRL | pkt2cmp.decode\_W\_control | Bins for all possible values including corner cases |
| 3. | MEM\_CNTRL | pkt2cmp.decode\_Mem\_control | Bins for all possible values including corner cases |
| 4. | PC1\_SEL | pcselect1 | Bins for all possible values including corner cases |
| 5. | ALUCNTRL\_SEL | alu\_control | Bins for all possible values including corner cases |

2. covergroup CONT\_INT\_SIG\_cg

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Cover Point Name | Values Used/Cov. Pts. Used | Coverage |
| 1. | BP\_ALU\_1 | pkt2cmp.control\_bypass\_alu\_1 | Bins for possible values of bypass alu1 {0,1} |
| 2. | BP\_ALU\_2 | pkt2cmp.control\_bypass\_alu\_2 | Bins for possible values of bypass alu2 {0,1} |
| 3. | BP\_MEM\_1 | pkt2cmp.control\_bypass\_mem\_1 | Bins for possible values of bypass mem1 {0,1} |
| 4. | BP\_MEM\_2 | pkt2cmp.control\_bypass\_mem\_2 | Bins for possible values of bypass mem2 {0,1} |
| 5. | MEM\_ST | pkt2cmp.control\_mem\_state | Bins for possible values of mem state {0,1,2,3} |
| 6. | BR\_TK | pkt2cmp.control\_br\_taken | Bins for branch taken or not taken |
| 7. | EN\_DEC | pkt2cmp.control\_enable\_decode | Bins for enable/disable decode stage |
| 8. | EN\_FET | pkt2cmp.control\_enable\_fetch | Bins for enable/disable fetch stage |
| 9. | EN\_UP | pkt2cmp.control\_enable\_updatePC | Bins for enable/disable PC Update |
| 10. | EN\_WB | pkt2cmp.control\_enable\_writeback | Bins for enable/disable writeback stage |
| 11. | EN\_EXE | pkt2cmp.control\_enable\_execute | Bins for enable/disable execute stage |

**3. Cover Properties Used**

The cover properties are used mainly for reset and control signal behaviors to assert the transitions. These can be found in test bench file. These include:

|  |  |  |
| --- | --- | --- |
| S.No. | Cover Property | Assertion Behavior |
| 1. | Reset | Cover property created to make sure that reset behavior is as expected and values are assigned to variable aptly. |
| 2. | Branch Taken | Cover property for JUMP instruction was created to make sure that branch is always taken |
| 3. | Dependencies like bypass alu1 & 2 | 2 cover properties were created for bypass alu1 and bypass alu2 to make sure that these are high when there is dependency |
| 4. | Load Store dependencies | 2 cover properties were created for bypass mem1 and bypass mem2 to make sure that these are high when there is dependency |
| 5. | Mem State Machine coverage | 7 cover properties were created to track the transitions of mem state from one to another. |

There are few assertions in the scoreboard which make sure that enable signals and outputs are transitioned correctly and were not included in the cover property to avoid redundancy.

**4. Directed Test cases**

Theses test cases can be found in Gen.v. The below scenarios were covered in directed test cases to ensure functional coverage:

1. Dependencies

Read After Write MEM dependency:

LD R1, A

ADD R2, R1, R2

Write After Write dependency:

ADD R1, R2, R3

ST R1, A

Read After Write ALU dependency:

ADD R1, R2, R3

AND R5,R1, R4

To make sure bypass alu and bypass mem values are getting generated by controller correctly, these instruction sets were given these instruction sets were given along with few more for source 2, as part of directed test cases.

2. Branch Taken condition

1. Branch if positive

ADD R0, R0, 10

ADD R1, R1, 10

ADD R2, R1, R0

ADD R4, R2, R1

ADD R5, R0, R0

BRA if positive to label\_1

ADD R0, R0, 10

ADD R1, R1, 10

2. Similarly branch if negative and branch if zero were tested for branch taken as 0 or 1.

3. Sequential cases for ALUóMEMORYóALUóCONTROL instructions

4. Corner cases for VSR1 and VSR2 (impacting aluin1 and aluin2) and hence initializing the registers to min and max values

There were around 50 directed test cases created to achieve the above conditions.

**5. Coverage Analysis**