# LC3 Microcontroller

# Coverage/Verification Plan

The objective of the project was to completely verify the LC3 microcontroller using a reusable and fully layered testbench for the verification.

**Verification Levels**- The first level of verification is to test each sub module of the LC3 in the order they appear in the pipeline starting from FETCH to the MEM\_ACCESS and lastly the CONTROLLER. Once blocks are individually verified the whole LC3 can be verified at a more abstract system level by monitoring the interactions between each block for random combinations of instructions.

**Required Tools**- Layered-testbench is implemented using System Verilog. The simulations tool used is Mentor Graphics QuestaSim.

**Verification Strategy**- System Verilog classes are used to create the transactions. These transaction processed by the transactors which are also implemented using System Verilog Classes. The transactions are communicated between the classes using the SV Mailboxes. The Mailboxes communication is used to sync up the Generator, Driver and Scoreboard. At the Receiver side the outputs of the DUT including the outputs of the internal blocks were collected in a similar mailbox and the Receiver sends it to the scoreboard to check the output received from DUT with the output of the golden model. The communication with the DUT is modeled in interfaces. There are two interfaces in the testbench “LC3\_top.if.sv” which communicates at the pin level of DUT and “LC3\_probe\_if.sv” which is used to probe the internal signals from the periphery of the individual blocks.

COVERGROUP DESCRIPTIONS

File-Name: /system\_tb/Coverage.sv

Covergroups are defines in the module named Coverage.sv. This module is instantiated in the “LC3.test\_top.sv”.

For coverage implementation we have followed the module based coverage approach. This approach provides more flexibility in controlling the triggers for the covergroups and makes the testbench modular.

1. ALU Instructions

1. This covergroup defines the coverage for the ALU instructions of LC3.
2. ALU\_OPR\_cg

Number of cover-points: 28

Coverpoint-Description:

*=> Cov\_alu\_opcode*: All the ALU operation i.e. ADD, AND and NOT.

*=> Cov\_imm\_en*: Immediate Enable(INSTR\_DOUT[5]).

*=> Cov\_SR1, Cov\_SR2, Cov\_DR*: All the values of SR1, SR2 and DR registers.

*=> Cov\_imm5*: All the values of imm5 qualified with the immediate type instructions.

*=> Xc\_opcode\_imm\_en*: Cross-Cover *Cov\_alu\_opcode & Cov\_imm\_en*.

*=> Xc\_opcode\_dr\_sr1\_imm5*: Cross-Cover *Cov\_alu\_opcode, Cov\_SR1, Cov\_DR & Cov\_imm5* for only immediate mode instructions.

*=> Xc\_opcode\_dr\_sr1\_sr2*: Cross-Cover *Cov\_alu\_opcode, Cov\_SR1, Cov\_DR & Cov\_imm5* for only register mode instructions.

*=> Cov\_aluin1, Cov\_aluin1\_corner, Cov\_aluin2, Cov\_aluin2\_corner*: ALU operand values ‘aluin1’ and ‘aluin2’ are binned into 8 bins. The \*\_corner covergroups covers all the corner case values of the ‘aluin1’ and ‘aluin2’ , which are binned based on the corner case like ‘ALL\_0’, ‘ALL\_1’ etc.

*=> Xc\_opcode\_aluin1, Xc\_opcode\_aluin2*: Cross-Cover *Cov\_alu\_opcode, Cov\_aluin1\_corner* to cover all the corner case values for each instruction. Similarly cross-cover for *Cov\_alu\_opcode, Cov\_aluin2\_corner.*

*=> Cov\_opr\_zero\_zero, Cov\_opr\_zero\_all1, Cov\_opr\_all1\_zero, Cov\_opr\_all1\_all1*: Different combination of corner scenarios ‘aluin1’ and ‘aluin2’ are covered for each of the opcode values. In these covergroups all the combination of ‘ALL\_0’ and ‘ALL\_1’ values are covered.

*=> Cov\_opr\_alt01\_alt01, Cov\_opr\_alt01\_alt10, Cov\_opr\_alt10\_alt01, Cov\_opr\_alt10\_alt10*: Different combination of corner scenarios ‘aluin1’ and ‘aluin2’ are covered for each of the opcode values. In these covergroups all the combination of alternate 1 and 0 are covered.

*=> Cov\_opr\_pos\_pos, Cov\_opr\_pos\_neg. Cov\_opr\_neg\_pos, Cov\_opr\_neg\_neg*: Different combination of corner scenarios ‘aluin1’ and ‘aluin2’ are covered for each of the opcode values. In these covergroups all the combination of positive and negative values of ‘aluin1’ and ‘aluin2’ are covered.

2. MEMORY Instructions

1. This covergroup defines the coverage for the Memmory operations of LC3.
2. MEM\_OPR\_cg

Number of cover-points: 9

Coverpoint-Description:

*=> Cov\_mem\_opcode*: All the Mem operation "LD, LDR, LDI, LEA, ST, STR, STI".

*=> Cov\_BaseR*: All the values of the BaseR, registers are covered and qualified wit the LDR and STR instructions.

*=> Cov\_SR*: All the values of the SR register are covered and qualified with the store instruction ST, STI and STR instructions.

*=> Cov\_DR*: All the values of the DR register are covered and qualified with the load instruction LD, LDI, LEA and LDR instructions.

*=> Cov\_PCoffset9, Cov\_PCoffset9\_c*: Values of the PCoffset9 binned to 8 bins and \*\_c coverpoint covers all the corner case values.

*=> Cov\_PCoffset6, Cov\_PCoffset6\_c*: Values of the PCoffset6 binned to 8 bins and \*\_c coverpoint covers all the corner case values.

*=> Xc\_BaseR\_DR\_offset6, Xc\_BaseR\_SR\_offset6*: Cross-Cover PCoffset6, SR, BaseR for STR and similarly PCoffset6, DR, BaseR for LDR.

3. CONTROL Instructions

1. This covergroup defines the coverage for the Control operations of LC3.

Coverage Group: CTRL\_OPR\_cg

Number of cover-points: 7

Coverpoint-Description:

*=> Cov\_ctrl\_opcode*: All the CNTRL operation "BR, JMP"".

*=> Cov\_BaseR*: All the values of the BaseR, registers are covered and qualified with the JMP instructions.

*=> Cov\_NZP, Cov\_PSR*: All the values of the NZP and PSR register are covered and qualified with the JMP instruction.

*=> Cov\_PCoffset9, Cov\_PCoffset9\_c*: Values of the PCoffset9 binned to 8 bins and \*\_c coverpoint covers all the corner case values.

*=> Xc\_NZP\_PSR*: Cross-Cover all the possible combinations of values of *Cov\_NZP, Cov\_PSR*.

4. Order of Instructions

1. This covergroup defines the coverage for all the possible combinations for the different type of instructions. This covers all the combination of [ALU] , [MEMORY] and [CNTRL] instruction and all the combination of [ALU ⬄ MEMORY ⬄ CONTROL] sequences of instructions.
2. OPR\_SEQ\_cg

Number of cover-points: 3

COVER-PROPERTY DESCRIPTIONS

File-Name: /system\_tb/LC3\_probe\_if.sv

Concurrent Assertions are coded in the probe interface file to verify the different scenarios and sequences of DUT mainly the controller. These scenarios are being covered to make sure these scenarios have been exercised by the testcases.

1. ‘reset’ Behaviour

1. The reset logic in LC3 is synchronous.

Cover the reset behavior of all the synchronous signals. Assertions are verifying this behavior in “LC3\_probe\_if.sv” and these assertions are covered using the cover property construct.

Cover-Property Name: LC3\_probe\_if.sv/\*\_rst

2. 'br\_taken’

1. Covers the conditions for the branch. The branch shall take place if any of the condition being sought (N, Z, or P is 1)

Cover-Property Name: CTRL\_br\_taken\_jmp

3. 'Enable\_\*’

1. Covers the assertion and de-assertion sequences of “enable\_decode”, “enable\_execute” and “enable\_writeback” based on the memory state.

Cover-Property Name: CTRL\_enable\_fetch, CTRL\_enable\_decode, CTRL\_enable\_mem\_state\_3, CTRL\_cmp\_instr\_rise, CTRL\_cmp\_instr\_fell

4. Dependencies of ALU ⬄ LOAD 'bypass\_alu\_1’ & ‘bypass\_alu\_2’

1. All the different scenarios of ALU⬄LOAD instruction are exercised.

Cover-Property Name: CTRL\_bypass\_alu\_1\_AA, CTRL\_bypass\_alu\_2\_AA, CTRL\_bypass\_alu\_1\_AS, CTRL\_bypass\_alu\_2\_AS

5. Dependencies of ALU ⬄ LOAD⬄STORE 'bypass\_mem\_1’ & ‘bypass\_mem\_2’

1. All the different scenarios of ALU⬄LOAD⬄STORE instruction are excercised.

Cover-Property Name: CTRL\_bypass\_mem\_1\_LA, CTRL\_bypass\_mem\_2\_LA, CTRL\_bypass\_mem\_1\_LS, CTRL\_bypass\_mem\_2\_LS

6. Memory State Machine coverage

1. All the transition memory state machine are covered.

Cover-Property Name: CTRL\_mem\_state\_3\_1, CTRL\_mem\_state\_3\_0, CTRL\_mem\_state\_3\_2, CTRL\_mem\_state\_2\_3, CTRL\_mem\_state\_1\_0, CTRL\_mem\_state\_1\_2, CTRL\_mem\_state\_0\_3

7. Memory State Machine flow for STI and LDI

1. All the transition memory state machine for STI and LDI are covered.

Cover-Property Name: CTRL\_mem\_state\_STI, CTRL\_mem\_state\_LDI

8. State Machine Stalling

1. Covers the conditions for the scenarios which stall the state machine.

Cover-Property Name: CTRL\_fsm\_stall

8. Writeback Enable after STORE and LOAD instruction

1. Covers the conditions for writeback enable after STORE and LOAD instructions.

Cover-Property Name: CTRL\_enable\_wb\_ST, CTRL\_enable\_wb\_LD