ECE 745 ASIC VERIFICATION

Fall 2018

LC3 MICROCONTROLLER VERIFICATION

COVERAGE REPORT

Ryan Geary
Tushar Trehon
Harish Jamakhandi
Mehrnaz Sadeghian

GROUP 9

Introduction:

The objective of this project was to implement a layered testbench using SystemVerilog that could achieve high coverage while remaining reusable. In order to achieve a layered design for our software, we implemented separate sections of our design using the object oriented capabilities that are built in to the SystemVerilog language. We were able to separate the the data and instruction generators of our code into an object that utilized constants in order to cover more possible operation scenarios. Whenever a new instruction and element of data was fed into the DUT, we launched a threaded checking architecture that would concurrently generate golden references for all the stages and check that reference against the actual outputs from the DUT. Once operations were completed for a cycle, all error metrics were reported back to a global scoreboard where they were stored and accumulated until the end of the simulation. This scoreboard proved useful when debugging a buggy version of the DUT, as it directly pointed us toward the stage that was generating the bug and how often the bug was being generated. Cover groups and assertions were also employed by this project aid in achieving higher amounts of coverage, and will be discussed later in this report.

Compilation/Running Instructions

1. If running this testbench on the clean DUT, ensure that all DUT files, module data_defs files, and the data_defs.vp file are located within a directory entitled LC3_cleanDUT which is within the partent directory.

   If running for a buggy DUT, make sure the data_defs file is located within the LC3_cleanDUT folder mentioned above. Rename the buggy _defs.vp file to data_defs.vp and rename the clean version of data_defs.vp to something else temporarily.

2. This submission contains a 'work' library; however, if you would like to create a new one, delete the existing one and use the command 'vlib work' with the ModelSim command prompt within the folder containing all the testbench source files.

3. Within the ModelSim command prompt, enter the command 'do compile.do'. If an error is thrown, it is probably due to the fact that you are not currently in the directory that holds this compile script. If so, navigate to the appropriate directory.

4. Allow the testbench to run to completion. We have structured the testbench within its source code to run the appropriate number of tests until coverage saturates.

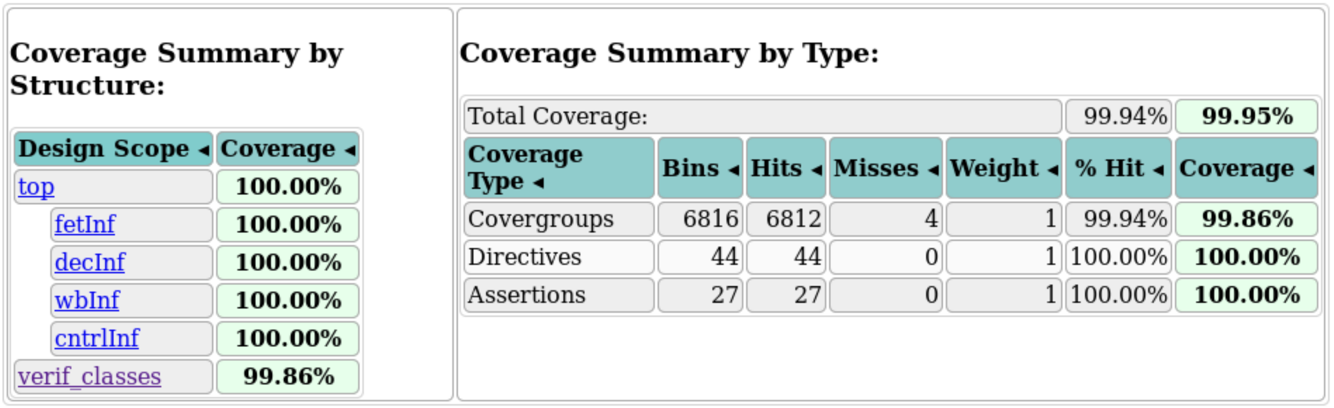
5. Once the run has completed, a scoreboard will be printed within the ModelSim command prompt that reports the number of tests ran, the number of failed test cases, as well as the stages in which those errors occurred.

6. After inspecting the scoreboard, you must generate a coverage report. First enter the command : 'coverage save coverage_report.ucdb'. This will create a file to report coverage. Next enter the command: 'vcover report -html -htmldir coverage_report -verbose coverage_report.ucdb' This will fill the report with coverage statistics from the last run.

7. To inspect this coverage report, within a file explorer, navigate to the coverage_report directory and double click on the covsummary.html file. This will open up your default browser and display all relevant coverage information.

Coverage Achieved:

Overall Coverage Achieved: 97.34.

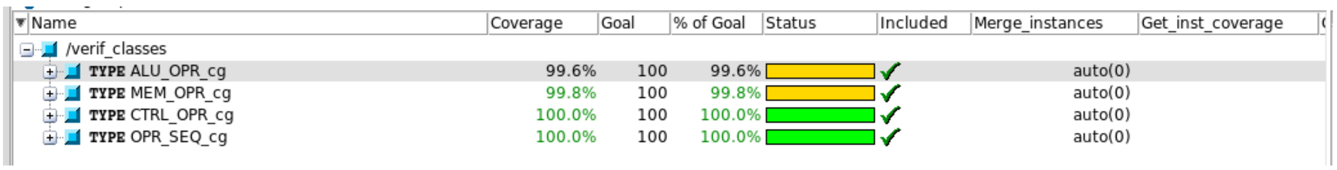


Coverage Summary by Structure:

Design Scope ◂	Coverage ◂
top	100.00%
fetInf	100.00%
decInf	100.00%
wbInf	100.00%
cntrlInf	100.00%
verif_classes	99.86%

Coverage Summary by Type:

Total Coverage:					99.94%	99.95%
Coverage Type ◂	**Bins ◂**	**Hits ◂**	**Misses ◂**	**Weight ◂**	**% Hit ◂**	**Coverage ◂**
Covergroups	6816	6812	4	1	99.94%	99.86%
Directives	44	44	0	1	100.00%	100.00%
Assertions	27	27	0	1	100.00%	100.00%



Name	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage
/verif_classes							
TYPE ALU_OPR_cg	99.6%	100	99.6%	🟨	✓	auto(0)	
TYPE MEM_OPR_cg	99.8%	100	99.8%	🟨	✓	auto(0)	
TYPE CTRL_OPR_cg	100.0%	100	100.0%	🟩	✓	auto(0)	
TYPE OPR_SEQ_cg	100.0%	100	100.0%	🟩	✓	auto(0)	

Cover Groups:

All Cover Groups were created, ALU_OPR_cg, MEM_OPR_cg, CTRL_OPR_cg, and OPR_SEQ_cg. In each of them different cover points were created. The definitions of all the following rules have been provide in Verification_plan.docx.

Cover Group	Cover Point	Cover Description
ALU_OPR_cg	Cov_alu_opcode	bins ADD_bin = {`ADD}; bins AND_bin = {`AND}; bins NOT_bin = {`NOT};
	Cov_ADD_AND	bins ADD_p = {`ADD}; bins AND_p = {`AND};
	Cov_NOT	bins NOT_p = {`NOT};

	Cov_imm_en	bins H = {1'b1};
		bins L = {1'b0};

	Cov_SR1	bins sr1_alu[] = {[7:0]};
	Cov_SR2	bins sr2_alu[] = {[7:0]};
	Cov_DR	bins dr_alu[] = {[7:0]};
	Cov_imm5	bins imm5_alu[] = {[5'b00000: 5'b11111]};
	Xc_opcode_imm_en	cross Cov_ADD_AND, Cov_imm_en;
	Xc_opcode_dr_sr1_imm5	cross Cov_ADD_AND, Cov_SR1, Cov_DR, Cov_imm5;
	Xc_opcode_dr_sr1_sr2	cross Cov_ADD_AND, Cov_SR1, Cov_DR, Cov_SR2 iff(dec.golden_ref.IR[5] == 1'b0);
	Xc_opcode_dr_src1_not	cross Cov_NOT,Cov_SR1,Cov_DR;
	Cov_aluin1	option.auto_bin_max = 8;

	Cov_aluin1_corner	bins aluin1_corner_high = {16'hffff};
		bins aluin1_corner_low= {16'h0000};

	Cov_aluin2	option.auto_bin_max = 8;

	Cov_aluin2_corner	bins aluin2_corner_high = {16'hffff};
		bins aluin2_corner_low= {16'h0000};

	Xc_opcode_aluin1	cross Cov_alu_opcode, Cov_aluin1_corner;
	Xc_opcode_aluin2	cross Cov_alu_opcode, Cov_aluin2_corner;
	Cov_aluin1_zero	bins aluin1_corner_low= {16'h0000};
	Cov_aluin1_one	bins aluin1_corner_high= {16'hffff};
	Cov_aluin2_zero	bins aluin2_corner_low= {16'h0000};
	Cov_aluin2_one	bins aluin2_corner_high= {16'hffff};
	Cov_aluin1_alt10	bins aluin1_alt_MSB0= {16'b0101010101010101};
	Cov_aluin1_alt01	bins aluin1_alt_MSB1= {16'b1010101010101010};
	Cov_aluin2_alt10	bins aluin2_alt_MSB0= {16'b0101010101010101};
	Cov_aluin2_alt01	bins aluin2_alt_MSB1= {16'b1010101010101010};
	Cov_aluin1_pos	bins aluin1_pos= {1'b0};
	Cov_aluin1_neg	bins aluin1_neg= {1'b1};
	Cov_aluin2_pos	bins aluin2_pos= {1'b0};

	Cov_aluin2_neg	bins aluin2_neg= {1'b1};
	Cov_opr_zero_zero	cross Cov_aluin1_zero, Cov_aluin2_zero,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_zero_all1	Cov_aluin1_zero, Cov_aluin2_one,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_all1_zero	cross Cov_aluin1_one, Cov_aluin2_zero,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_all1_all1	cross Cov_aluin1_one, Cov_aluin2_one,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_alt01_alt01	cross Cov_aluin1_alt01, Cov_aluin2_alt01,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_alt01_alt10	cross Cov_aluin1_alt01, Cov_aluin2_alt10,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_alt10_alt01	cross Cov_aluin1_alt10, Cov_aluin2_alt01,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_alt10_alt10	cross Cov_aluin1_alt10, Cov_aluin2_alt10,Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_pos_pos	cross Cov_aluin1_pos, Cov_aluin2_pos, Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_pos_neg	cross Cov_aluin1_pos, Cov_aluin2_neg, Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};
	Cov_opr_neg_pos	cross Cov_aluin1_neg, Cov_aluin2_pos, Cov_alu_opcode{ ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
	Cov_opr_neg_neg	cross Cov_aluin1_neg, Cov_aluin2_neg, Cov_alu_opcode{

		ignore_bins others = binsof (Cov_alu_opcode) intersect {`NOT};}
MEM_OPR_cg	Cov_mem_opcode	bins LD_bins = {`LD}; bins LDR_bins = {`LDR}; bins LEA_bins = {`LEA}; bins LDI_bins = {`LDI}; bins ST_bins = {`ST}; bins STR_bins = {`STR}; bins STI_bins = {`STI};
	Cov_BaseR_LDR	bins LDR_base[] ={[3'b000:3'b111]};
	Cov_BaseR_STR	bins STR_base[] ={[3'b000:3'b111]};
	Cov_SR_ST	bins ST_source[] ={[3'b000:3'b111]};
	Cov_SR_STI	bins STI_source[] ={[3'b000:3'b111]};
	Cov_SR_STR	bins STR_source[] ={[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_SR_LDI	bins LDI_DR[] ={[3'b000:3'b111]};
	Cov_SR_LEA	bins LEA_DR[] ={[3'b000:3'b111]};
	Cov_PCoffset9	option.auto_bin_max = 8; sim:/top/#INITIAL#147
	Cov_PCoffset6	option.auto_bin_max = 8;

	Cov_PCoffset9_c	bins offset9_low_corner = {9'b000000000};
		bins offset9_high_corner = {9'b111111111};

	Cov_PCoffset6_c	bins offset9_low_corner = {6'b000000};
		bins offset9_high_corner = {6'b111111};

	Xc_BaseR_SR_offset6	cross Cov_PCoffset6, Cov_SR_STR, Cov_BaseR_STR, Cov_mem_opcode{
		ignore_bins others = binsof (Cov_mem_opcode) intersect {`LD, `LDR, `LDI, `LEA, `ST, `STI};}

	Xc_BaseR_DR_offset6	Xc_BaseR_DR_offset6: cross Cov_PCoffset6, Cov_DR_LDR, Cov_BaseR_LDR, Cov_mem_opcode{
		ignore_bins others = binsof (Cov_mem_opcode) intersect {`LD, `STR, `LDI, `LEA, `ST, `STI};}

	Xc_BaseR_SR_offset6_c	cross Cov_PCoffset6_c, Cov_SR_STR, Cov_BaseR_STR, Cov_mem_opcode{ ignore_bins others = binsof (Cov_mem_opcode) intersect {`LD, `LDR, `LDI, `LEA, `ST, `STI};}
	Xc_BaseR_DR_offset6_c	Xc_BaseR_DR_offset6_c: cross Cov_PCoffset6_c, Cov_DR_LDR, Cov_BaseR_LDR, Cov_mem_opcode{ ignore_bins others = binsof (Cov_mem_opcode) intersect {`LD, `STR, `LDI, `LEA, `ST, `STI};}
CTRL_OPR_cg	Cov_ctrl_opcode	bins BR_bins = {`BR}; bins JMP_bins = {`JMP};
	Cov_BaseR	bins JMP_BaseR[] = {[7:0]};
	Cov_NZP	bins JMP_NZP[] = {7};
	Cov_PSR	bins JMP_PSR4 = {3'b100}; bins JMP_PSR2 = {3'b010}; bins JMP_PSR1 = {3'b001};
	Cov_PCoffset9	option.auto_bin_max = 8;
	Cov_PCoffset9_c	bins offset9_low_corner = {9'b000000000}; bins offset9_high_corner = {9'b111111111};
	Xc_NZP_PSR	cross Cov_NZP, Cov_PSR;
OPR_SEQ_cg	Cov_opcode_order	bins ALU_Memory = (`AND, `ADD, `NOT => `LD, `LDR, `LDI, `LEA, `ST, `STR, `STI); bins Memory_ALU = (`LD, `LDR, `LDI, `LEA,`ST, `STR, `STI => `AND, `ADD, `NOT); bins ALU_Control = (`AND, `ADD, `NOT => `BR, `JMP ); bins Control_ALU = ( `BR, `JMP => `AND, `ADD, `NOT); bins Memory_Control = (`LD, `LDR, `LDI, `LEA, `ST, `STR, `STI => `BR, `JMP); bins Control_Memory = ( `BR, `JMP => `LD, `LDR, `LDI, `LEA, `ST, `STR, `STI);
Cov_opcode_ADD_others		//ADD --> ALU Operation

		bins ADD_AND = (`ADD => `AND);
		bins ADD_NOT = (`ADD => `NOT);
		bins ADD_ADD = (`ADD => `ADD);
		
		//ADD --> Load Instruction
		bins ADD_LD = (`ADD => `LD);
		bins ADD_LDR = (`ADD => `LDR);
		bins ADD_LDI = (`ADD=> `LDI);
		bins ADD_LEA = (`ADD => `LEA);
		
		//ADD --> Store Instruction
		bins ADD_ST = (`ADD => `ST);
		bins ADD_STR = (`ADD => `STR);
		bins ADD_STI = (`ADD => `STI);
		
		//ADD --> Control Instruction
		bins ADD_BR = (`ADD => `BR);
		bins ADD_JMP = (`ADD => `JMP);

Cov_opcode_AND_others		//AND --> ALU Operation
		bins AND_ADD = (`AND => `ADD);
		bins AND_AND = (`AND => `AND);
		bins AND_NOT = (`AND => `NOT);
		
		//AND --> Load Instruction
		bins AND_LD = (`AND => `LD);
		bins AND_LDR = (`AND => `LDR);

		bins AND_LDI = (`AND => `LDI);
		bins AND_LEA = (`AND => `LEA);
		
		//And --> Store Instruction
		bins AND_ST = (`AND => `ST);
		bins AND_STR = (`AND => `STR);
		bins AND_STI = (`AND => `STI);
		
		//ADD --> Control Instruction
		bins AND_BR = (`AND => `BR);
		bins AND_JMP = (`AND => `JMP);

Cov_opcode_NOT_others		//NOT --> ALU Operation
		bins NOT_ADD = (`NOT => `ADD);
		bins NOT_AND = (`NOT => `AND);
		bins NOT_NOT = (`NOT => `NOT);
		
		//NOT --> Load Instruction
		bins NOT_LD = (`NOT => `LD);
		bins NOT_LDR = (`NOT => `LDR);
		bins NOT_LDI = (`NOT => `LDI);
		bins NOT_LEA = (`NOT => `LEA);
		
		//NOT --> Store Instruction
		bins NOT_ST = (`NOT => `ST);
		bins NOT_STR = (`NOT => `STR);
		bins NOT_STI = (`NOT => `STI);

		//NOT --> Control Instruction
		bins NOT_BR = (`NOT => `BR);
		bins NOT_JMP = (`NOT => `JMP);

cov_opcode_LD_others		//LD--> ALU Operation
		bins LD_ADD = (`LD => `ADD);
		bins LD_AND = (`LD => `AND);
		bins LD_NOT = (`LD => `NOT);
		
		//LD --> Control Instruction
		bins LD_BR = (`LD => `BR);
		bins LD_JMP = (`LD => `JMP)

Cov_opcode_LDR_others		//LDR--> ALU Operation
		bins LDR_ADD = (`LDR => `ADD);
		bins LDR_AND = (`LDR => `AND);
		bins LDR_NOT = (`LDR => `NOT);
		
		//LDR --> Control Instruction
		bins LDR_BR = (`LDR => `BR);
		bins LDR_JMP = (`LDR => `JMP);

Cov_opcode_LDI_others		//LDI--> ALU Operation
		bins LDI_ADD = (`LDI => `ADD);
		bins LDI_AND = (`LDI => `AND);
		bins LDI_NOT = (`LDI => `NOT);
		
		//LDI --> Control Instruction
		bins LDI_BR = (`LDI => `BR);

		bins LDI_JMP = (`LDI => `JMP);

Cov_opcode_ST_others		//ST--> ALU Operation
		bins ST_ADD = (`ST => `ADD);
		bins ST_AND = (`ST => `AND);
		bins ST_NOT = (`ST => `NOT);
		
		//ST --> Control Instruction
		bins ST_BR = (`ST => `BR);
		bins ST_JMP = (`ST => `JMP);

Cov_opcode_STR_others		//STR--> ALU Operation
		bins STR_ADD = (`STR => `ADD);
		bins STR_AND = (`STR => `AND);
		bins STR_NOT = (`STR => `NOT);
		
		//STR --> Control Instruction
		bins STR_BR = (`STR => `BR);
		bins STR_JMP = (`STR => `JMP);

Cov_opcode_STI_others		//STI--> ALU Operation
		bins STI_ADD = (`STI => `ADD);
		bins STI_AND = (`STI => `AND);
		bins STI_NOT = (`STI => `NOT);
		
		//STI--> ALU Operation
		bins STI_BR = (`STI => `BR);
		bins STI_JMP = (`STI => `JMP);

Cov_opcode_BR_others		//BR--> ALU Operation
		bins BR_ADD = (`BR => `ADD);
		bins BR_AND = (`BR => `AND);

		bins BR_NOT = (`BR => `NOT);
		
		//BR --> Load Instruction
		bins BR_LDR = (`BR => `LDR);
		bins BR_LDI = (`BR => `LDI);
		bins BR_LEA = (`BR => `LEA);
		
		//BR --> Store Instruction
		bins BR_ST = (`BR => `ST);
		bins BR_STR = (`BR => `STR);
		bins BR_STI = (`BR => `STI);

Cov_opcode_JMP_others		//JMP--> ALU Operation
		bins JMP_ADD = (`JMP => `ADD);
		bins JMP_AND = (`JMP => `AND);
		bins JMP_NOT = (`JMP => `NOT);
		//JMP--> ALU Operation
		bins JMP_ADD = (`JMP => `ADD);
		bins JMP_AND = (`JMP => `AND);
		bins JMP_NOT = (`JMP => `NOT);
		
		//JMP --> Load Instruction
		bins JMP_LDR = (`JMP => `LDR);
		bins JMP_LDI = (`JMP => `LDI);
		bins JMP_LEA = (`JMP => `LEA);
		
		//JMP --> Store Instruction
		bins JMP_ST = (`JMP => `ST);
		bins JMP_STR = (`JMP => `STR);

		bins JMP_STI = (`JMP => `STI);
		//JMP --> Load Instruction
		bins JMP_LDR = (`JMP => `LDR);
		bins JMP_LDI = (`JMP => `LDI);
		bins JMP_LEA = (`JMP => `LEA);
		
		//JMP --> Store Instruction
		bins JMP_ST = (`JMP => `ST);
		bins JMP_STR = (`JMP => `STR);
		bins JMP_STI = (`JMP => `STI);

ASSERTIONS:

1. **Reset**

   i) Fetch
```
        property lC3_rst_if;
           reset |-> ##1 (pc == 16'h3000);
        endproperty
```
   ii) Decode
```
        property lc3_rst_dc;
           reset                      |->                      ##1
     (!((Mem_Control)||(W_Control)||(E_Control)||(IR)||(npc_out))
     );
        endproperty
```
   iii) Execute
```
        property lC3_rst_ex;
           @(posedge clock)
           reset                      |->                      ##1
        (!((Mem_Control_out)||(W_Control_out)||(dr)||(NZP)||(IR_E
        xec)||(aluout)||(pcout)||(M_Data)));
        endproperty
```
   iv) Write back
```
           property lC3_rst_wb;
              @(posedge clock)
              reset |-> ##1 (!psr);
           endproperty
```
2. **Branch Taken**
```
 property CTRL_br_taken_jmp;
    @(posedge clock)
    |(NZP & psr) |-> br_taken;
 Endproperty
```

3. **Enable fetch**
```
 property CTRL_enable_fetch_low;
    @(posedge clock)
    (IR[15:12] == `ST || IR[15:12] == `STR  || IR[15:12] == `STI
 || IR[15:12] == `LD || IR[15:12] == `LDR || IR[15:12] == `LDI ||
 IMem_dout[15:12]   ==   `BR   ||   IMem_dout[15:12]   ==   `JMP)
 |=>  !enable_fetch;
 endproperty

 property CTRL_enable_fetch_high1;
    @(posedge clock)
    (IR[15:12] == `ST || IR[15:12] == `STR || IR[15:12] == `LD ||
 IR[15:12] == `LDR ) |=>  ##1 enable_fetch;
 endproperty

 property CTRL_enable_fetch_high2;
    @(posedge clock)
    (IR[15:12]  ==  `LDI  ||  IR[15:12]  ==  `STI  )  |=>    ##2
 enable_fetch;
 Endproperty

 property CTRL_enable_fetch_high3;
    @(posedge clock)
    (IMem_dout[15:12] == `BR) || (IMem_dout[15:12] == `JMP) |=>
 ##3 enable_fetch;
 endproperty
```

4. **Decode enable**

```
property CTRL_enable_decode_low1;
  @(posedge clock)
  (IR[15:12] == `ST || IR[15:12] == `STR  || IR[15:12] == `STI || IR[15:12] == `LD || IR[15:12] == `LDR || IR[15:12]
== `LDI ) |=>  !enable_decode;
endproperty

property CTRL_enable_decode_low2;
  @(posedge clock)
  (IMem_dout[15:12] == `BR || IMem_dout[15:12] == `JMP)  |=>  ##1 !enable_decode;
endproperty

property CTRL_enable_decode_high1;
  @(posedge clock)
  (IR[15:12] == `ST || IR[15:12] == `STR || IR[15:12] == `LD || IR[15:12] == `LDR ) |=>  ##1 enable_decode;
Endproperty

property CTRL_enable_decode_high2;
  @(posedge clock)
  (IR[15:12] == `LDI || IR[15:12] == `STI ) |=>  ##2 enable_decode;
Endproperty

property CTRL_enable_decode_high3;
  @(posedge clock)
  (IMem_dout[15:12] == `BR) || (IMem_dout[15:12] == `JMP) |=>  ##4 enable_decode;
Endproperty
```

5. **Execute enable**

```
property CTRL_enable_execute_low1;
 @(posedge clock)
 (IR[15:12] == `ST || IR[15:12] == `STR  || IR[15:12] == `STI ||
IR[15:12] == `LD || IR[15:12] == `LDR || IR[15:12] == `LDI )
|=>  !enable_execute;
endproperty

property CTRL_enable_execute_low2;
 @(posedge clock)
 (IMem_dout[15:12] == `BR || IMem_dout[15:12] == `JMP) |=>
##2 !enable_execute;
endproperty

property CTRL_enable_execute_high1;
 @(posedge clock)
 (IR[15:12] == `ST || IR[15:12] == `STR || IR[15:12] == `LD ||
IR[15:12] == `LDR ) |=>  ##1 enable_execute;
Endproperty

property CTRL_enable_execute_high2;
 @(posedge clock)
 (IR[15:12] == `LDI || IR[15:12] == `STI ) |=>  ##2 enable_execute;
Endproperty

property CTRL_enable_execute_high3;
 @(posedge clock)
 ((IMem_dout[15:12] == `BR) || (IMem_dout[15:12] == `JMP)) |=>
##5 enable_execute;
endproperty
```

6. **Write Back enable:**

```
 property CTRL_enable_writeback_Load;
    @(posedge clock)
    (IR[15:12] == `LD || IR[15:12] == `LDR ) |=>  ##1 enable_writeback;
 endproperty


 property CTRL_enable_writeback_Brn;
       @(posedge clock)
       (IMem_dout[15:12] == `BR) || (IMem_dout[15:12] == `JMP) |=>  ##6
 enable_writeback;
 endproperty


 property CTRL_enable_wb_ST1;
    @(posedge clock)
    (IR_Exec[15:12]==`STR       ||       IR_Exec[15:12]==`ST       ||
    IR_Exec[15:12]==`STI) |-> enable_writeback==1'b0;
 endproperty

 property CTRL_enable_wb_ST2;
    @(posedge clock)
    (IR_Exec[15:12]==`STR      ||      IR_Exec[15:12]==`ST)      |=>
 enable_writeback==1'b1;
 endproperty

 property CTRL_enable_wb_ST3;
    @(posedge clock)
    (IR_Exec[15:12]==`STI) ##3 enable_writeback==1'b1;
 Endproperty
```

7. **Bypass ALU**

```
        property CTRL_bypass_alu_1_AA;
```

```
       @(posedge clock)
    ((IR[15:12] == `ADD) || (IR[15:12] == `AND) || (IR[15:12] == `NOT))
&&  ((IR_Exec[15:12]  ==  `ADD)  ||  (IR_Exec[15:12]  ==  `AND)  ||
(IR_Exec[15:12]  ==  `NOT)  /*||  (IR_Exec[15:12]  ==  `LEA)*/)  &&
(IR_Exec[11:9] == IR[8:6]) |-> bypass_alu_1 == 1'b1;
    endproperty

    property CTRL_bypass_alu_2_AA;
       @(posedge clock)
    ((IR[15:12] == `ADD) || (IR[15:12] == `AND) || (IR[15:12] == `NOT))
    &&  ((IR_Exec[15:12]  ==  `ADD)  ||  (IR_Exec[15:12]  ==  `AND)  ||
(IR_Exec[15:12] == `NOT)/*|| (IR_Exec[15:12] == `LEA)*/)
    && ((IR_Exec[11:9] == IR[2:0]) && (IR[5] == 1'b0)) |-> bypass_alu_2
== 1'b1;
    endproperty


    property CTRL_bypass_alu_1_AS;
       @(posedge clock)
    ((IR_Exec[15:12]  ==  `ADD)  ||  (IR_Exec[15:12]  ==  `AND)  ||
(IR_Exec[15:12] == `NOT))
      && ((IR[15:12] == `STR))
      && (IR_Exec[11:9] == IR[8:6]) |-> bypass_alu_1 == 1'b1;
    endproperty


    property CTRL_bypass_alu_2_AS;
    @(posedge clock)
    ((IR_Exec[15:12]  ==  `ADD)  ||  (IR_Exec[15:12]  ==  `AND)  ||
(IR_Exec[15:12] == `NOT))
       &&  ((IR[15:12]  ==  `STR)||(IR[15:12]  ==  `ST)||(IR[15:12]  ==
`STI))
     && (IR_Exec[11:9] == IR[11:9]) |-> bypass_alu_2 == 1'b1;
    endproperty


    property CTRL_bypass_alu_1_AL;
    @(posedge clock)
    ((IR_Exec[15:12]  ==  `ADD)  ||  (IR_Exec[15:12]  ==  `AND)  ||
(IR_Exec[15:12] == `NOT)) && (IR[15:12] == `LDR) && (IR_Exec[11:9] ==
IR[8:6]) |-> bypass_alu_1 == 1'b1;
    endproperty
```

8. Bypass memory line assertions

```
   property CTRL_bypass_mem_1_LA;
       @(posedge clock)
    ((IR_Exec[15:12] == `LD) || (IR_Exec[15:12] == `LDR) || (IR_Exec[15:12]
    == `LDI)) && ((IR[15:12] == `ADD) || (IR[15:12] == `AND) || (IR[15:12]
    == `NOT)) && (IR_Exec[11:9] == IR[8:6]) |-> bypass_mem_1 == 1'b1;
    endproperty

    property CTRL_bypass_mem_2_LA;
       @(posedge clock)
            ((IR_Exec[15:12]  ==  `LD)  ||  (IR_Exec[15:12]  ==  `LDR)  ||
    (IR_Exec[15:12]  ==  `LDI))  &&  ((IR[15:12]  ==  `ADD)  ||  (IR[15:12]  ==
    `AND) || (IR[15:12] == `NOT)) && ((IR_Exec[11:9] == IR[2:0]) && (IR[5]
    == 1'b0)) |-> bypass_mem_2 == 1'b1;
    endproperty
```

9. Mem_State Transitions

```
property CTRL_mem_state_LDI;
  @(posedge clock)
     (IR[15:12] == `LDI) |=>  mem_state == 2'b01 ##1 mem_state == 2'b00
 ##1 mem_state == 2'b11;
 endproperty

 property CTRL_mem_state_STI;
 @(posedge clock)
       (IR[15:12] == `STI) |=>  mem_state == 2'b01 ##1 mem_state == 2'b10
 ##1 mem_state == 2'b11;
 endproperty

 property CTRL_enable_mem_state_ST_STR;
 @(posedge clock)
    (IR[15:12] == `ST || IR[15:12] == `STR ) |=>  mem_state == 2'b10 ##1
 mem_state == 2'b11;
 endproperty


 property CTRL_enable_mem_state_STI_LDI;
 @(posedge clock)
       (IR[15:12] == `LDI || IR[15:12] == `STI ) |=>  mem_state == 2'b01
 ##2 mem_state == 2'b11;
 endproperty

 property CTRL_enable_mem_state_LD_LDR;
    @(posedge clock)
          (IR[15:12] == `LD || IR[15:12] == `LDR ) |=>  mem_state == 2'b00
 ##1 mem_state == 2'b11;
 endproperty
```

```
property CTRL_mem_state_3_1;
@(posedge clock)
       (mem_state == 2'b11) && ((IR[15:12] == `LDI) || (IR[15:12] == `LDI))
|=>  mem_state == 2'b01;
endproperty

property CTRL_mem_state_3_0;
@(posedge clock)
       (mem_state == 2'b11) && (IR[15:12] == `LDI) |=>  mem_state == 2'b01
##1 mem_state == 2'b00;
endproperty


property CTRL_mem_state_3_2;
@(posedge clock)
       (mem_state == 2'b11) && (IR[15:12] == `STI) |=>  mem_state == 2'b01
##1 mem_state == 2'b10;
endproperty


property CTRL_mem_state_2_3;
@(posedge clock)
(mem_state == 2'b10) && (complete_data == 1) |=>  mem_state == 2'b11;
endproperty

property CTRL_mem_state_0_3;
@(posedge clock)
(mem_state == 2'b00) && (complete_data == 1) |=>  mem_state == 2'b11;
endproperty


property CTRL_mem_state_1_2;
@(posedge clock)
(mem_state == 2'b01) && (IR_Exec[15:12] == `STI) |=>  mem_state == 2'b10;
endproperty

property CTRL_mem_state_1_0;
@(posedge clock)
(mem_state == 2'b01) && (IR_Exec[15:12] == `LDI) |=>  mem_state == 2'b00;
endproperty

property CTRL_enable_writeback_STR_fall1;
@(posedge clock)
   (IR[15:12] == `ST || IR[15:12] == `STR || IR[15:12] == `LD || IR[15:12]
== `LDR || IR[15:12] == `LDI || IR[15:12] == `STI ) |=>  !enable_writeback;
endproperty

property CTRL_enable_writeback_BR_fall2;
@(posedge clock)
        (IMem_dout[15:12]  ==  `BR  ||  IMem_dout[15:12]  ==  `JMP)  |=>
##2 !enable_writeback;

endproperty
```