ECE 499

Formal Property Verification

Group 1: Ming Chu Chiu

Bolivar Beleno

Feb 23 2024

**I) Create Assumptions**

Assumptions can be found in properties.sv file.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement # | Requirement Description | ID | Property Name |
| 7 | req[M1] is asserted for exactly one cycle | 11 | req1\_held\_one\_cycle |
| req[M2] is asserted for exactly one cycle | req2\_held\_one\_cycle |
| req[M3] is asserted for exactly one cycle | req3\_held\_one\_cycle |
| 16 | At no point can req[n] and done[n] be both asserted. | 42 | p16\_never\_doneN\_and\_reqN |
| 15 | Each request line is asserted for one clk cycle only, not held. | 71 | p15\_done1\_held\_one\_cycle |
| p15\_done2\_held\_one\_cycle |
| p15\_done3\_held\_one\_cycle |
| Helper | eventually will trigger req1 | N/A | pXX\_req1\_eventually |
| eventually will trigger req2 | pXX\_req2\_eventually |
| eventually will trigger req3 | pXX\_req3\_eventually |
| eventually will trigger done1 | pXX\_req1\_eventually |
| eventually will trigger done2 | pXX\_done2\_eventually |
| eventually will trigger done3 | pXX\_done3\_eventually |
| Helper | eventually will trigger req2 & req3 | N/A | pXX\_req2\_and\_req3\_eventually |

**II) Create cover properties**

Covers can be found in properties.sv file.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement # | Requirement Description | ID | Property Name |
| 15 | hen module1 releases memory, assert its done line | 38 | p15\_m1\_done\_when\_release |
| hen module2 releases memory, assert its done line | 39 | p15\_m2\_done\_when\_release |
| hen module3 releases memory, assert its done line | 40 | p15\_m3\_done\_when\_release |
|  |  |  |  |

**III) Test Case Coding**

N/A

**IV) Run the formal Tool**

All properties satisfactorily passed assertions.

**V) Explain <100% coverage**

Our group experienced some issues when trying to define properties regarding events that occurred between positive clock edges. Amongst the most common issues encountered were counter examples that exercised properties at infinity, or more precisely, never asserted certain inputs disprove properties. In those cases, it was reasonable to include assumptions that at some time, these inputs will be satisfied. For example, when testing the alternating characteristic for request M2 and M3, the counter example would be to never request M2 and M3 simultaneously.

**VI) Scripting**

Executable can be found as: “runme.sh” file. And for formal, run Jasper Gold and execute inside the command line “include runme.tcl”.

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type of Coverage** | **Point of Interest/Requirement** | **Percentage Covered [%]** | **Notes** |
| Code Coverage | Branch Coverage | 100 | N/A |
|  | FSM State Coverage | 100 | N/A |
|  | Statement Coverage | 100 | N/A |
|  | 5.[ [cg\_tc\_reset\_in\_all\_states] For all possible states when rst is triggered the Module goes to initial state | 100 | Test case went smooth |
| Functional Coverage | 8.[fb\_M1\_can\_interrupt]  assert that M1 can interrupt M2 and M3 | 100 | Test case went smooth |
|  | 24. [fb\_M2\_and\_M3\_cannot\_interrupt] for each module on first cycle, iff not done[self] and req[!self] illegal bin if transitions to any possible state that can interrupt | 100 | Cases where M1 and either M2 or M3 requested were hard to reach at low input of random variables. Very high chance to optimize syntax as all conditions were atomized and is convoluted |
|  | 36.[ fb\_smooth\_transition\_no\_idle] forbidden bin on transition from any module to IDLE [accmodule] if done[self] and req[!self] on same clock | 100 | Test case went smooth |
| SVA (Functional Coverage Backup for FPV) | 3, 4, 7, 8, 9,10, 13, 14, 15, 16, 19 | 100 | Assertions fired but resolved, found bugs during. |

**Bug Tracker**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Description | signals | status | Comments | File | Discovering Test ID |
| 1 | asynchronous reset not working | rst | fixed | line 46 | controller\_wrong.sv |  |
| 2 | M3 doesn't get interrupt by M1 | [m3in\_2p -> m1it\_2p] | fixed | line [119~128] [M3in\_2p -> M1it\_2p] | controller\_wrong.sv |  |
| 3 | M2sd\_2p doesn't go into IDLE (loop) | m2sd\_2p -> IIDLE\_2p | fixed | line 179 | controller\_wrong.sv |  |
| 4 | nb\_interrupt never goes to zero | nb\_interrupt | fixed | line 50 and [55~58] | controller\_wrong.sv |  |
| 5 | memory hogging | req[n] == done[n] == 1 | fixed | line 133, etc | controller\_wrong.sv |  |
| 6 | M3 req not satisfied when accmodule==2 and M2 is done | ns | fixed | added don't cares '?' to line 96, 114, 225, 244 to satisfy the appropriate requests | controller\_wrong.sv | 35 |
| 7 | M1in\_3p/2p cannot smooth transition | M1in\_3p -> (M2in\_3p or M3in\_3p) | fixed | line 212 and 83 | controller\_wrong.sv | 35 |
| 8 | M2sd\_2p/3p retriggering after smooth ending [conflict with done and req] | M2sd\_2p/3p ->M2in\_2p/3p | fixed | line 184, etc | controller\_wrong.sv | 43 |
| 8 | M3sd\_2p/3p retriggering after smooth ending [conflict with done and req] | M3sd\_2p/3p ->M3in\_2p/3p | fixed | line 333, 196 | controller\_wrong.sv | 43 |
| 9 | M1sd\_2p/3p memory hogging (repeated request) | M1sd\_2p/3p ->M1in\_2p/3p | fixed | line 171, 311 | controller\_wrong.sv | 68 |

**Iteration Random Inputs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of random input iteration (length = 100 clocks) | FSM state [%] | FSM transition [%] | Statement [%] | Branch [%] | Total Cover group [%] |
| 3 | 94.73 | 64.86 | 82.66 | 80.13 | 85.79 |
| 10 | 100 | 84.68 | 94.66 | 92.05 | 95.83 |
| 100 | 100 | 100 | 100 | 100 | 100 |

**Test Plan**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Requirement | Verification Technique | Verification Constructs | Objects in Verilog | Test Cases | Comments |
| 0 | 1. File controller.sv | None | Visual Inspection | N/A | N/A | Currently called controller\_wrong.sv |
| 1 | 2. Moore FSM implementation | Linting | [p2\_MooreFSM] Case Statement implemented for FSM next state logic | N/A | N/A | N/A |
| 2 |  | Code Coverage | FSM code coverage: ps ns | ps, ns | State visits, Transitions | N/A |
| 3 | 3. One-hot encoding implementation | Linting | [p3\_oneHot\_encoding] States are enumerated in one-hot style Case expression is 1'b1 and case items in FSM case statement is one bit of present state | N/A | N/A | N/A |
| 65 |  | Functional Coverage Backup(SVA) | [fb\_oneHot\_encoding] check if one hot (state 0 inclusive) | ps | [tc\_oneHot\_encoding] |  |
| 70 |  | FPV |  |  |  |  |
| 4 | 4. Active high asynchronous reset | FPV | [p4\_active\_high\_async\_reset] assert that on the positive edge of '**reset**' signal, **mstate** goes to IDLE, **accmodule** is set to 0, and **nb\_interrupt** is set to 0. | mstate, accmodule, nb\_interrupt | N/A (FPV) | Confirm with customer if the nb\_interrupt value needs to be zeroed upon reset. |
| 66 |  | Functional Coverage Backup(SVA) | [p4\_active\_high\_async\_reset] assert that on the positive edge of '**reset**' signal, **mstate** goes to IDLE, **accmodule** is set to 0, and **nb\_interrupt** is set to 0. | mstate, accmodule, nb\_interrupt | [tc\_reset\_in\_all\_cases] ensure that reset is asserted in all the states there are. |  |
| ~~5~~ |  | ~~Functional Coverage Backup (CG)~~ | ~~[fb\_active\_high\_async\_reset] Illegal bin~~ **~~mstate~~** ~~!= IDLE,~~ **~~accmodule~~** ~~!=0,~~ **~~nb\_interrupt~~** ~~!=0 for the~~ **~~clk~~** ~~cycle after reset is deasserted.~~ | ~~mstate, accmodule, nb\_interrupt~~ | ~~[tc\_active\_high\_async\_reset] assert 'reset' input and then deassert it. We expect mstate==IDLE, accmodule==0, and nb\_interrupt==0 for the clk cycle after 'reset' is deasserted.~~ | Unable to check the correctness using just functional coverage. When we used sampling signal of @(posedge rst) to check the values of present state, it's giving us the present state of the pre-region (prior value, when rst is still low) not the value after posedge goes high. If we sample @(negedge rst) that would not be accurate either, since rst is active high. We decided to check the correctness of this function using FPV (SVA). However, we could use functional coverage to ensure that the test case happens (rst is asserted in different states.) |
| 5 |  | Functional Coverage (ensure test case) | [cg\_reset\_in\_all\_states] coverpoint mstate @posedge rst | reset, mstate | [tc\_reset\_in\_all\_cases] ensure that reset is asserted in all the states there are. | N/A |
| 6 | 4. Reset called *reset* | Linting | [p4\_reset\_signal\_name]  Active high asynchronous reset named '**reset**' | reset | N/A | N/A |
| 7 | 5. M1 can interrupt M2 and M3 | FPV | [p5\_M1\_can\_interrupt] assert that when M1 request on **clk** posedge, it gets the access to memory (**accmodule** ==1) on that **clk** edge. | clk, req, accmodule | N/A (FPV) | This property really checks for a not done transition but under assumption we can copnlcude that this property is sufficient. In case we want to be verbose with the property we can impolement it similar to ID15 |
| 8 |  | Functional Coverage (backup) | [cg\_M1\_can\_interrupt] assert that M1 can interrupt M2 and M3 | clk, req, mstate | [tc\_M1\_can\_interrupt] test that M1 can interrupt via req[MONE] | To achieve 100% coverage modification to the supplied design was submitted (see bugTracker.txt) |
| 9 | 6.Request array called req with req[2] for M3, req[1] for M2 and req[0] for M1. | Linting | [p6] check for **req[2:0]** declaration | req | N/A | N/A |
| 10 | 6.req[2] for M3, req[1] for M2 and req[0] for M1 | Code coverage | req[1],req[2] and req[3] | req | req = 3'b001, req = 3'b010;, req = 3'b100; | Line 40 of tb.sv |
| 11 | 7. Each request line is asserted for one clk cycle only, not held. | FPV | [req1\_held\_one\_cycle] assert **req** rise, followed by one **clk** cycle, and deasserted. [req2\_held\_one\_cycle] assert req rise, followed by one clk cycle, and deasserted.  [req3\_held\_one\_cycle] assert req rise, followed by one clk cycle, and deasserted. | req | N/A (FPV) | N/A |
| 64 |  | Functional Coverage Backup(SVA) | [fb\_reqN\_held\_one\_cycle] request line is held for one cycle |  |  |  |
| 12 | 8. Requests are evaluated and acted upon on *clk* posedge | FPV | [p8\_req\_action] assert that **accmodule** transition only happens when **req**uest is high upon posedge **clk** | clk, req, accmodule | N/A (FPV) | Should we be checking mstate as well, or just accmodule would suffice? |
| 67 |  | Functional Coverage (backup) | [p8\_req\_action] assert that **accmodule** transition only happens when **req**uest is high upon posedge **clk** | clk, req, accmodule |  |  |
| ~~13~~ |  | ~~FPV~~ | ~~[p8\_unsuccessful\_req] assert that no transition occurs on~~ **~~accmodule~~** ~~when~~ **~~req~~**~~uest is high but no posedge~~ **~~clk~~** | ~~clk, req, accmodule~~ | ~~N/A (FPV)~~ | ~~Should we be checking mstate as well, or just accmodule would suffice?~~ |
| ~~14~~ |  | ~~Functional Coverage (backup)~~ | ~~[fb\_unsuccessful\_req] coverpoint = accmodule @(req[M1]) and iff (!posedge clk) illegal bins bad\_req\_m3\_m1 = 3 => 1  illegal bins bad\_req\_m2\_m1 = 2 => 1  illegal bins bad\_req\_m1 = 0 => 1   coverpoint = accmodule @(req[M2]) and iff (!posedge clk) illegal bins bad\_req\_m3\_m2 = 3 => 2 (can happen on smooth transition) illegal bins bad\_req\_m1\_m2 = 1 => 2 illegal bins bad\_req\_m2 = 0 => 2  coverpoint = accmodule @(req[M3]) and iff (!posedge clk) illegal bins bad\_req\_m3\_m3 = 3 => 3 (can happen on smooth transition) illegal bins bad\_req\_m1\_m3 = 1 => 3 illegal bins bad\_req\_m3 = 0 => 3~~ | ~~clk, req, accmodule~~ | ~~[tc\_unsuccessful\_req] assert req[M1] during a period where there is no posedge of clk.  assert req[M2] during a period where there is no posedge of clk.  assert req[M3] during a period where there is no posedge of clk.~~ | ~~How do we express "not posedge clk"?~~ The planned sampling event is req, which will not allow us to see transition of accmodule upon posedge of the clk. We decided to not use functional coverage for this requirement because for ensuring the test case that req is asserted not on the clock edge, we would need !posedge clk event, which we couldn't find. |
| 15 | 9. If M1 interrupted, can hold memory for at most 2 cycles. | FPV | [p9\_M1\_interrupt\_limit] assert that accmodule != 1 after two clk cycles if M1 requested during M2 and M3 access. | clk, req, accmodule, mstate | N/A (FPV) | N/A |
| 68 |  | Functional Coverage Backup (SVA) | [p9\_M1\_interrupt\_limit] assert that accmodule != 1 after two clk cycles if M1 requested during M2 and M3 access. | clk, req, accmodule, mstate |  |  |
| 16 |  | Functional Coverage (ensure test case) | [cg\_M1\_interrupt\_limit] coverpoint mstate @ posedge clk; transition FROM M2in\_2p = 4 TO M1id\_2p = 10  transition FROM M2in\_3p = 5 TO M1id\_3p = 11 transition FROM M3in\_2p = 6 TO M1id\_2p = 10  transition FROM M3in\_3p = 7 TO M1id\_3p = 11 | clk, mstate | [tc\_M1\_interrupt\_limit] ensure mstate is in M2 and M3 states and then transition into M1 interrupt state | Is it legal to cross twice in the same cover group? |
| 17 | 9. If M1 did not interrupt, can hold memory indefinitely | FPV | [p9\_M1\_infinite\_idle] assert that when accmodule 0 =>1, accmodule stays 1 if not done[M1] & clk. | clk, accmodule, done[M1] | N/A (FPV) | N/A |
| ~~18~~ |  | ~~FPV~~ | ~~[p9\_M1\_infinite\_M3done] assert that when (accmodule 3 =>1)&&(done[M3]), accmodule stays 1 if not done[M1] & clk.~~ | ~~clk, accmodule, done[M2]~~ | ~~N/A (FPV)~~ | ~~N/A~~ |
| 19 | 9.[M1id\_2p] is triggered | Code coverage | M1id\_2p | ps[M1id\_2p] | ps[M1id\_2p] actuated | Line 911 of tb.sv |
| ~~20~~ |  | ~~FPV~~ | ~~[p9\_M1\_infinite\_M2done] assert that when (accmodule 2 =>1)&&(done[M2]), accmodule stays 1 if not done[M1] & clk.~~ | ~~clk, accmodule, done[M3]~~ | ~~N/A (FPV)~~ | ~~N/A~~ |
| 21 |  | Functional Coverage | [p9\_ensure\_M1\_acc] cm1\_acc: coverpoint = req[M1] bins = {1}; cmi: coverpoint accmodule bins idle= {0}; cm2: coverpoint accmodule bins m2= {2}; cm3: coverpoint accmodule bins m3= {3}; cm2done: coverpoint done[M2] bin ={1}; cm3done: coverpoint done[M3] bin ={1}; cross cm1\_acc, cmi cross cm1\_acc, cm2, cm2done cross cm1\_acc, cm3, cm3done | clk, accmodule, done[1:0], req[M1] | [tc\_ ensure\_M1\_acc] req[M1] from IDLE, from M2 with done[M2], from M3 with done[M3]. | N/A |
| 22 | 10. M2 and M3 cannot interrupt any module | FPV | [p10\_M2\_cannot\_int]  assume req[M2] assert that (accmodule!=2) upon posedge clk if other module is accessing the memory (accmodule !=2) and not done. | clk, accmodule, done[1:0], req[M2] | N/A (FPV) | N/A |
| 23 |  | Functional Coverage Backup(SVA) | [fb\_M2\_didnt\_interrupt\_M1] property defines non interruption smooth transitions [fb\_M2\_didnt\_interrupt\_M3] property defines non interruption smooth transitions [fb\_M2\_didnt\_interrupt\_M1] property defines non interruption smooth transitions [fb\_M2\_didnt\_interrupt\_M1] property defines non interruption smooth transitions | clk, accmodule, done[1:0], req[M3] | [tc\_M2M3\_didnt\_interrupt] | N/A |
| 24 |  | Functional Coverage | [cg\_M2\_and\_M3\_cannot\_interrupt] for each module on first cycle, iff not done[self] and req[!self] illegal bin if transitions to any possible state that can interrupt | clk, ps, done, req, ns | [tc\_M2\_and\_M3\_cannot\_interrupt] req[M2] when ps is m3 or m1 and not done. req[M3] when ps is m2 or m1 and not done. | Not required, easily satisfiable |
| 25 | 11. Module is granted access only at positive edge of the clock | FPV | [p11\_acc\_gnt\_on\_pClk] assert that transition of **accmodule** only happens on posedge **clk** | clk, accmodule | N/A (FPV) | Consider alternative method to satisfy transition only at posedge clk |
| 26 | 12. Incase of M2 and M3 simultaneous *req,* access is alternated | FPV | [p12\_M23\_tiebreak] assume accmodule==0 or (1 & ~~done[M1]~~) or (mstate==m1sd\_2p or m1\_sd3p) assert that when **(req[M2] & req[M3])** on clk posedge, accmodule =>2 =>3 =>2 =>3 =>2. | clk, accmodule, done[1:0], req[1:0] | N/A (FPV) | Cannot assert req[M2]& req[M3] when done[M2] or done[M3] by requirement 16. |
| 27 |  | Functional Coverage | [p12\_ensure\_M23\_tiebreak] cm1done: coverpoint done[M1] bin = {1}; cm1sd: coverpoint mstate bin b\_m1sd [] = {12,13} cross creq2, creq3, cmi cross creq2, creq3, acc1, cm1done cross creq2, creq3, cm1sd | clk, accmodule, done[1:0], req[1:0], mstate | [tc\_ensure\_M23\_tiebreak] req[M2]&req[M3] when accmodule is 0, (1 & done[M1]), (1&& (mstate==12 or 13)) | N/A |
| 28 | 13. M2 and M3 can hold memory at most 2 *clk* cycles | FPV | [p13\_M2\_cycle\_limit]  assert if (accmodule ==2) then two clk cycles later (accmodule!=2) | clk, accmodule | N/A (FPV) | N/A |
| 29 |  | Functional Coverage Backup(SVA) | [fb\_M2\_cycle\_limit] M2 and M3 have two-cycle limit [fb\_M3\_cycle\_limit] M2 and M3 have two-cycle limit assert if (accmodule ==3) then two clk cycles later (accmodule!=3) | clk, accmodule | N/A (FPV) | N/A |
| 30 | 14. When a module asserts its done signal, another module may assert its request signal in that same clock cycle and receives access without a gap in-between. | FPV | [p14\_M1\_is\_done\_and\_M2\_requests] assume property M1 is done and M2 requests [p14\_M1\_done\_to\_M2\_no\_idle] assert property when "p14\_M1\_is\_done\_and\_M2\_requests" transition from M1 to M2 [p14\_M1\_is\_done\_and\_M3\_requests] assume property M1 is done and M3 requests [p14\_M1\_done\_to\_M3\_no\_idle] assert property when "p14\_M1\_is\_done\_and\_M3\_requests" transition from M1 to M3 | done[0] && req[1] M1in\_2p, M1it\_2p, M1id\_2p -> M2in\_2p M1in\_3p, M1it\_3p, M1id\_3p -> M2in\_3p done[0] && req[2] M1in\_2p, M1it\_2p, M1id\_2p -> M3in\_2p M1in\_3p, M1it\_3p, M1id\_3p -> M3in\_3p" | None - FPV | This could be very wrong because I may not be considering the right sequence |
| ~~31~~ |  | ~~FPV~~ |  |  |  | combined with above |
| 32 |  | FPV | [p14\_M2\_is\_done\_and\_M1\_requests] assume property M2 is done and M1 requests [p14\_M2\_done\_to\_M1\_no\_idle] assert property when "p14\_M2\_is\_done\_and\_M1\_requests" transition from M2 to M1 [p14\_M2\_is\_done\_and\_M3\_requests] assume property M2 is done and M3 requests [p14\_M2\_done\_to\_M3\_no\_idle] assert property when "p14\_M2\_is\_done\_and\_M3\_requests" transition from M2 to M3 | done[1] && req[0] M2in\_2p, M2it\_2p, M2id\_2p -> M1in\_2p M2in\_3p, M2it\_3p, M2id\_3p -> M1in\_3p done[1] && req[2] M2in\_2p, M2it\_2p, M2id\_2p -> M3in\_2p M2in\_3p, M2it\_3p, M2id\_3p -> M3in\_3p" | None - FPV | Same here |
| ~~33~~ |  | ~~FPV~~ |  |  | ~~None - FPV~~ | combined with above |
| 34 |  | FPV | [p14\_M3\_is\_done\_and\_M1\_requests] assume property M3 is done and M1 requests [p14\_M3\_done\_to\_M1\_no\_idle] assert property when "p14\_M3\_is\_done\_and\_M1\_requests" transition from M3 to M1 | done[2] && req[0] M3in\_2p, M3it\_2p, M3id\_2p -> M1in\_2p M3in\_3p, M3it\_3p, M3id\_3p -> M1in\_3p | None - FPV | Same here |
| 35 |  | Functional Coverage Backup(SVA) | [fb\_M1\_done\_to\_MN\_no\_idle]smooth transition from M1 to M2\_or\_M3 [fb\_M2\_done\_to\_MN\_no\_idle]smooth transition from M1 to M2\_or\_M3 [fb\_M3\_done\_to\_MN\_no\_idle]smooth transition from M1 to M2\_or\_M3 | accmodule, done && req | None - FPV |  |
| 36 |  | Functional Coverage | [cg\_smooth\_transition\_no\_idle] forbidden bin on transition from any module to IDLE [accmodule] if done[self] and req[!self] | [M1,M2 -> M3], [M1,M3 -> M2] and [M2,M3 -> M1] | [tc\_smooth\_transition\_no\_idle] if M[N] is done and any other module is requested at same cycle, guarantees a smooth transition | Maybe done[], req[M1 and MN] could be challenging due to the random nature of the request |
| 37 | 15. Each module has their own done signal | Linting | [p15\_done\_sig\_for\_each\_module] check for 'done [1:0]' declaration | N/A | N/A | N/A |
| 38 | 15. When it’s done using the memory, or it reaches the 2 clk cycles limit, it must assert its done for a cycle | FPV | [p15\_m1\_done\_when\_release] assume that if (mstate==10 or 11), until done[M1] ~~eventually~~ happens | mstate, done[M1] | N/A (FPV) | Team has decided that FPV is a better approach |
| 39 |  | FPV | [p15\_m2\_done\_when\_release] assume that if (accmodule==2), after 2 clk cycles, done[M2] is 1. | clk, accmodule, done[M2] | N/A (FPV) | N/A |
| 40 |  | FPV | [p15\_m3\_done\_when\_release] assume that if (accmodule==3), after 2 clk cycles, done[M3] is 1. | clk, accmodule, done[M3] | N/A (FPV) | N/A |
| ~~41~~ |  | ~~FPV~~ | ~~[p15\_done\_m1\_int] assume that if (mstate==12 or 13), after 2 clk cycles, done[M1] is 1.~~ | ~~clk, mstate, done[M1]~~ | ~~N/A (FPV)~~ | ~~N/A~~ |
| 71 |  | FPV | [p15\_done1\_held\_one\_cycle] assume that done line for each module is held for one clk cycle. [p15\_done2\_held\_one\_cycle]  [p15\_done3\_held\_one\_cycle] |  |  |  |
| 69 |  | Functional Coverage Backup(SVA) | [fb\_done1\_held\_one\_cycle] [fb\_done2\_held\_one\_cycle] [fb\_done3\_held\_one\_cycle] assert that done line for each module is held for one clk cycle. | done | Randomization | Note: For now, just 1 cycle done contraint |
| 42 | 16. Module cannot simultaneously assert its done and req lines | FPV | [p16\_never\_doneN\_and\_reqN] assume property never req and done |  | N/A (FPV) | Assume |
| 43 |  | Functional Coverage Backup(SVA) | [fb\_never\_doneN\_and\_reqN] never req and done | done[2:0], req[2:0] | [tc\_never\_doneN\_and\_reqN] | check design line 133, M1 Hogging memory is invalid per REQ16 (done[n] != req[n] if == 1) [m!it\_2p] |
| 44 | 17. Done signal called *done* for each module | Linting | [p17\_done\_early] assert property module transitions before two cycles after request iff done triggered |  |  | Maybe Linting? |
| 45 | 18. Cycle limit for M1, M2 and M3 | FPV | [p18\_M1\_no\_interrupt] assume M1 did not interrupt another module [p18\_M1\_M2\_M3\_cycle\_limit] assert property M1,M2, and M3 are held for at most two cycles if [p18\_M1\_interrupt] |  |  | See 12 13 and 17 |
| ~~46~~ |  | ~~FPV~~ | ~~[p18\_M1\_no\_interrupt\_unlimited\_cycles] assert property never step out of M1\_interrupt if not done M1~~ |  |  |  |
| 47 | 19. Memory is granted only if req is asserted. | FPV | [p19\_grant\_M1\_iff\_requested\_M1] assert property request M1 iff requested one cycle before | accmodule, req[M1] | N/A (FPV) |  |
| 48 |  | FPV | [p19\_grant\_M2\_iff\_requested\_M2] assert property request M2 iff requested one cycle before | accmodule, req[M2] | N/A (FPV) |  |
| 49 |  | FPV | [p19\_grant\_M3\_iff\_requested\_M3] assert property request M3 iff requested one cycle before | accmodule, req[M3] | N/A (FPV) |  |
| 50 |  | Functional Coverage Backup(SVA) | [fb\_grant\_M1\_iff\_requested\_M1] M1 iff requested one cycle before | accmodule, req[M1] | [tc\_grant\_M1\_iff\_requested\_M1] |  |
| 51 |  | Functional Coverage Backup(SVA) | [fb\_grant\_M2\_iff\_requested\_M2]M2 iff requested one cycle before | accmodule, req[M2] | [tc\_grant\_M2\_iff\_requested\_M2] |  |
| 52 |  | Functional Coverage Backup(SVA) | [fb\_grant\_M3\_iff\_requested\_M3] M3 iff requested one cycle before | accmodule, req[M3] | [tc\_grant\_M3\_iff\_requested\_M2] |  |
| 53 | 20. Inputs: req lines [*req*], done lines[*done*], reset line [*rst*], needed clocks [*clk*] | Linting | req, done, and reset covered in entry #8 and #33 check for 'clk' signal | N/A | N/A | How to lint? |
| 54 | 21. Output: Bus *mstate* which indicates what state the FSM | Linting | check for 'mstate' signal | N/A | N/A | How to lint? |
| 55 | 21. Output: bus *accmodule.* Indicates which module holds memory | Linting | check for 'accmodule' signal | N/A | N/A | How to lint? |
| 56 | 21. Output: 32 bit integer nb\_interrupts. Counts M2 and M3 interruptions. | Linting | check for 'nb\_interrupts' integer | N/A | N/A | How to lint? |
| 57 | 22. Uses enums to code states | Linting | check for 'enum' keyword | N/A | N/A | How to lint? |
| 58 | 23. Use case, casez, unique, priority when possible | Linting | check for 'case', 'casez', 'unique', 'priority' keywords | N/A | N/A | How to lint? |
| 59 | 24. Use parameters to index into done and req arrays | Linting | check for 'parameter' keyword | N/A | N/A | How to lint? |
| 60 | 25. Use testbench, tb() | None | Visual Inspection | None | None | None |
| 61 | 26. Use shell script, runme.sh to compile and run in cmd-line mode | None | Visual Inspection | None | None | None |
| 62 | 27. Comment state transitions | Linting | check for comments | N/A | N/A | How to lint? |
| 63 | 28. Confirm file naming convention as expressed in this test plan | None | Visual Inspection | None | None | None |