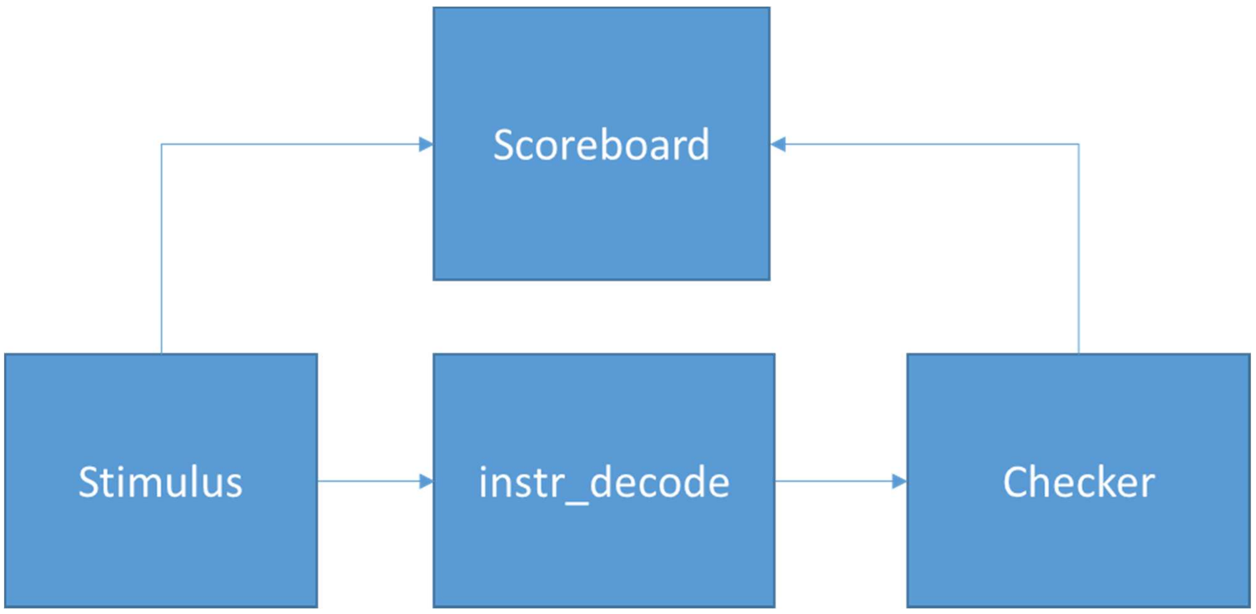
# ECE510: Final Project Proposal

## Unit-level Verification Strategy for ***instr_decode*** module

### Block Diagram



### Stimulus Generation

1. This module generates instructions mimicking the *memory_pdp* module.
2. For initial testing, the stimulus module will generate all instruction types that are supported by the *instr_decode* module.
3. The instructions supported by *instr_decode* module are: AND, TAD, ISZ, DCA, PMS, JMP, NOP, IAC, RAL, RTL, RAR, RTR, CML, CMA, CIA, CLA, CLA1, CLA_CLL, HLT, OSR, SKP, SNL, SZL, SZA, SNA, SMA, SPA, CLA2.
4. Once we generate all these instructions, the stimulus module then switches over to randomized testing by generating random values and loading random instructions from an array of instructions mentioned above.

### Checker

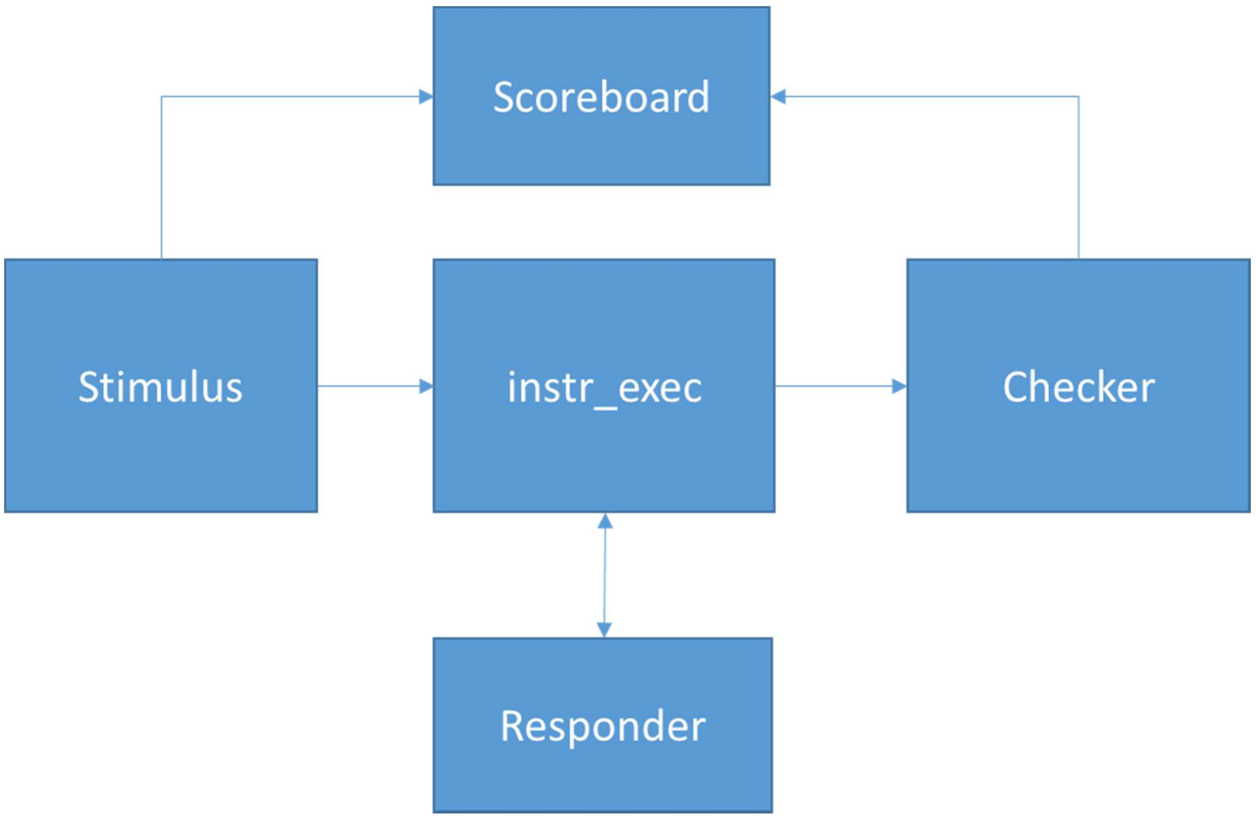
1. The checker module will verify the functional correctness by taking in inputs from the Stimulus module, computing the desired output and comparing it with the output generated by the *instr_decode* module.
2. The checker will have SystemVerilog Assertions to ensure valid state transitions.

Scoreboard

1. The scoreboard is our way of ensuring ***optimal coverage***.
2. The scoreboard will keep track of the type of and number of instructions that are generated by the stimulus
3. The scoreboard will keep track of the states being entered and we can ensure that all states were entered and covered by our stimulus
4. The scoreboard will keep track of the functionally correct and incorrect values generated by the stimulus and output by the *instr_decode* module

## Unit-level Verification Strategy for ***instr_decode*** module

### Block Diagram



Stimulus Generation

1. The stimulus will generate the inputs to the *instr_exec* module in the same format as generated by the *instr_decode* module.
2. As in the case of *instr_decode* module, we will first generate all the instructions supported by the *instr_exec* module for as part of our directed testing strategy.

3. We will then move on to randomized testing where we will generate random input values based on an array of supported instructions.
4. The instructions supported by the *instr_exec* module are: CLA_CLL, TAD, AND, ISZ, DCA, JMS, JMP, NOP

Checker

1. The checker will take in inputs from the stimulus module and compute the expected output values from the *instr_exec* module, which is then compared with the actual output from the *instr_exec* module to ensure functional correctness.
2. The checker will also have SystemVerilog Assertions to ensure valid state transitions.

Scoreboard

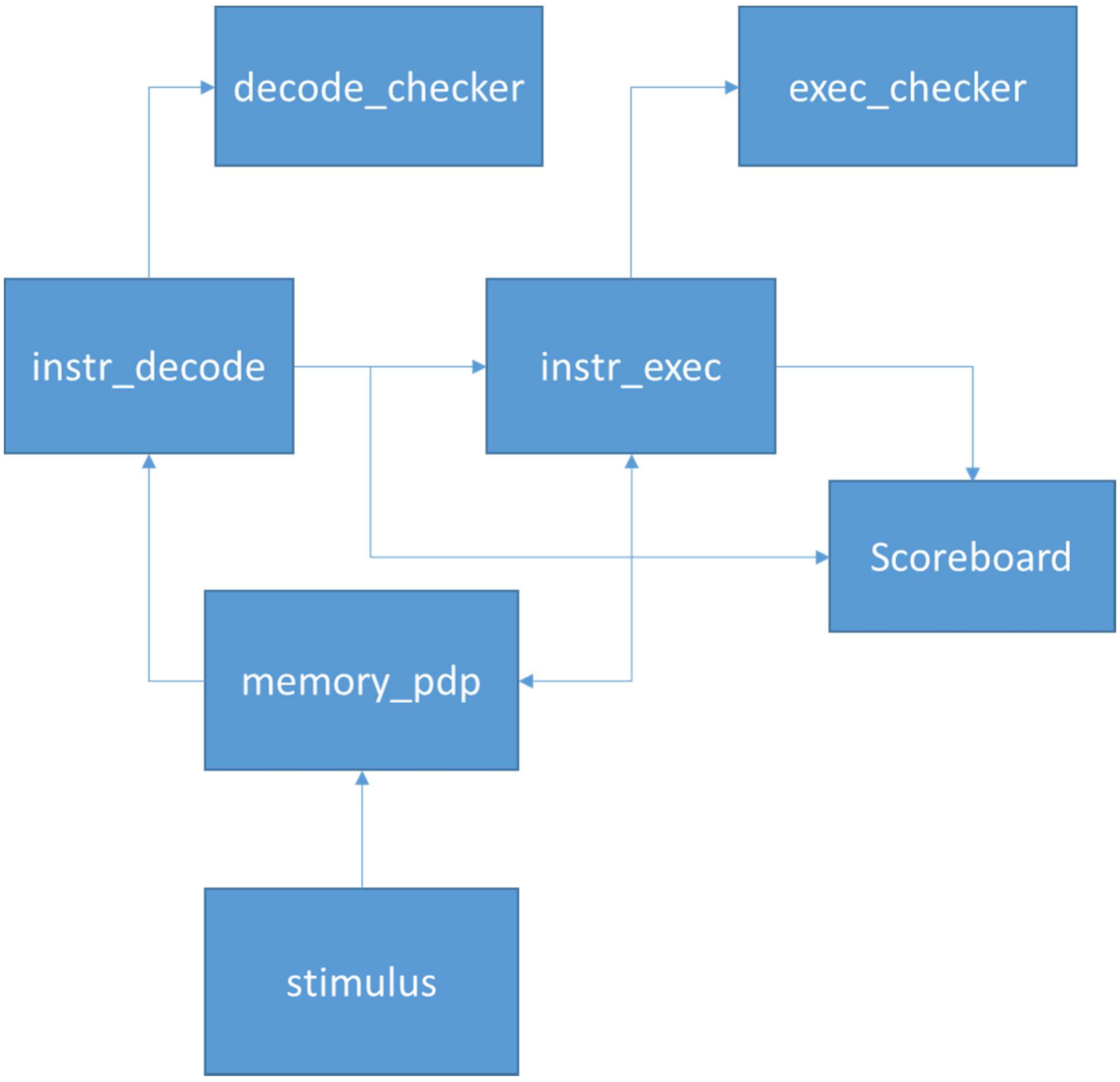
1. The scoreboard is our way of ensuring ***optimal coverage***.
2. The scoreboard will keep track of the type of and number of instructions that are generated by the stimulus
3. The scoreboard will keep track of the states being entered and we can ensure that all states were entered and covered by our stimulus
4. The scoreboard will keep track of the functionally correct and incorrect values generated by the stimulus and output by the *instr_exec* module

Responder

1. The responder module mimics the *memory_pdp* module and provides input data to the *instr_exec* module and also enables mimicking the *instr_exec* to write output data to the memory.

## Chip-level Verification Strategy for full-design

Block Diagram



Stimulus Generation

1. For stimulus generation, we have two phases
2. Phase 1; We hand code a bunch of instructions and run the simulation
3. Phase 2: We generate random instructions and run the simulation
4. In both cases the instructions are loaded into the *test.mem* file which serves as stimulus to the design

Checker

1. For Phase 1, we use the re-use the same checkers from Unit-level testing of *instr_exec* and *instr_decode* modules with minor modification, if required, to fit the full Chip-level verification framework
2. For Phase 2, if required, we build a 3rd checker which will use these older checkers to provide on-the-fly checking capabilities
3. So, in addition to checker capabilities described in our Unit-level verification strategy, we will also have real-time checking.

Scoreboard

1. As in the case of unit-level testing, the scoreboard keeps track of the type of and number of instructions being executed which will ensure ***optimal coverage***
2. The scoreboard will keep track of the states being entered and we can ensure that all states were entered and covered by our stimulus.
3. The scoreboard will keep track of the functionally correct and incorrect values generated by the stimulus and output by the full-design which will help us debug failures and narrow down the bug suspects and see patterns in bugs.