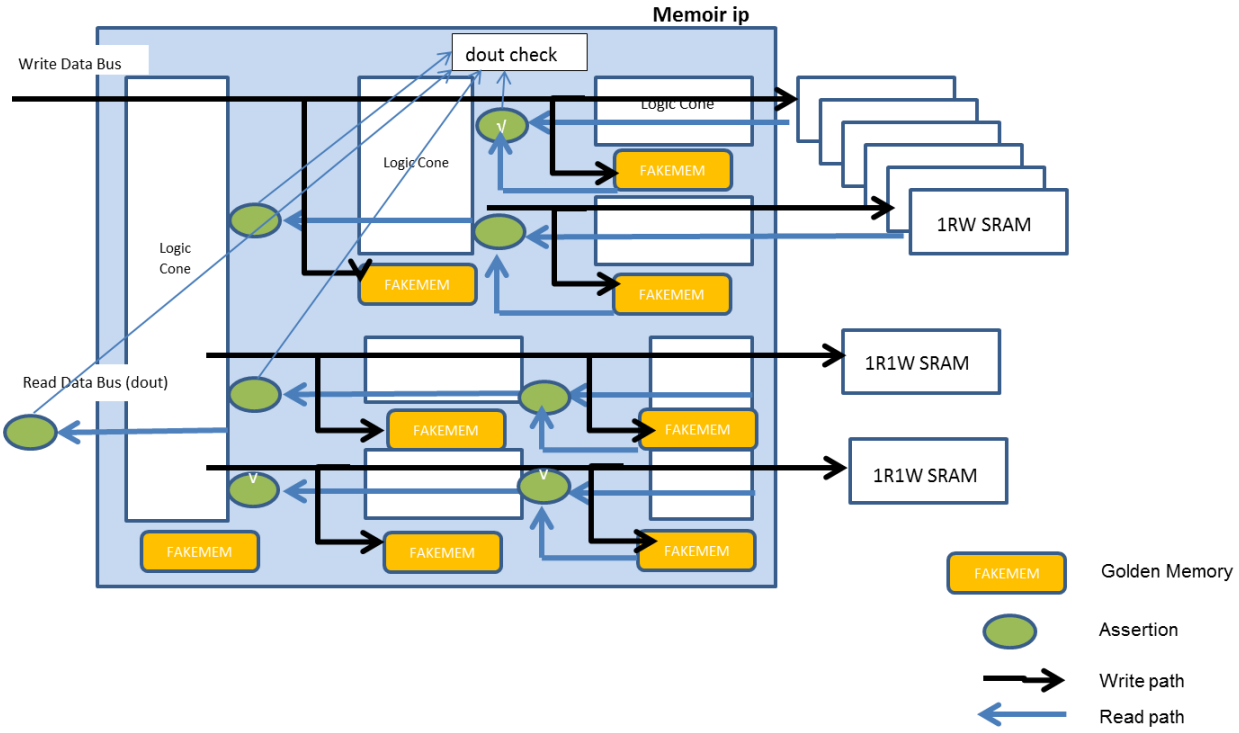
MEMOIR SYSTEMS

Formal Verification Overview – Memoir 1R1W IP

This document outlines the various approaches we used for formally verifying our design completely and thoroughly.



The diagram above shows the logical view of our approach to formally verify our design completely and thoroughly. We break the complete IP into several logic cones. Each cone has a corresponding golden memory model (fakemem). When a write request is launched at the input of any logic cone, it gets captured in the golden memory. On a read, we compare the read data returned by the logic cone with golden memory. The read check happens on the interface of the cone of logic. This check is captured in the form of an SVA assertion (i.e., a property) and we use an industry standard formal verification tool (IFV from cadence) to prove the assertions. Our assertions cover the complete code, including the logic inside each cone and all the interfaces between the cones.

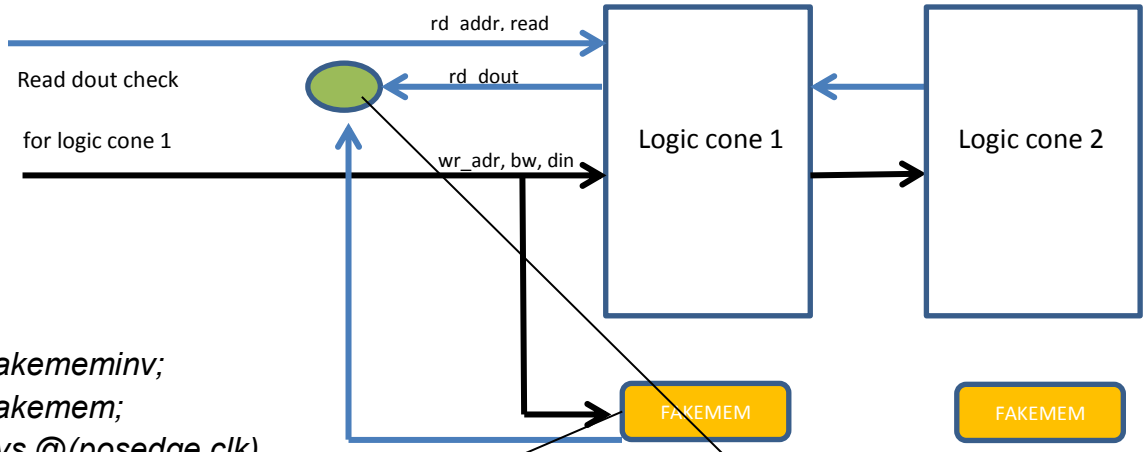
We have an in-house post processing flow to evaluate the formal coverage of the design. A script parses through the code for each IP being delivered and retrieves all the required checks in the design. Then it parses all run logs to prepare a matrix of all the properties that

have been proven. We use this matrix to qualify a design being 100% formally proven. The report generated by the script is attached in Appendix A.

By definition, the formal verification methodology provides exhaustive verification, if it converges for all the properties defined. As opposed to a traditional verification methodology, which provides only a specific set of test vectors, the formal tool applies all possible test vectors -- and under all these patterns the properties for all cones of logic in the design are proven.

We have used several design-for-verification techniques to guarantee that formal tool is able to converge the proofs completely on our IPs. The goal of these techniques is to reduce the state space both temporally and spatially.

The diagram below zooms into the formal verification infrastructure for a typical logic cone. It shows the code for the golden memory model and the property (SVA assertion) for a read dout check for a generic logic cone.




```
reg fakememinv;
reg fakemem;
always @(posedge clk)
  if (rst)
    fakememinv <= 1'b1;
  else if (write && (wr_adr == select_addr) ) begin
    fakememinv <= 1'b0;
    fakemem <= din[select_bit];
```

```
assertrd_dout_check: assert property (@(posedge clk) disable iff (rst) (read && (rd_adr
== select_addr)) |-> ##(MEM_DELAY)
                        (fakememinv || (rd_dout[select_bit] == fakemem)));
```

Appendix A

```
[O]: cmdline: $IFV_HOME/verilogparser/bin/fchkr.pl -name
memoir_1r1w_65536x72
[O]: reading $IFV_HOME/cutinfo/memoir_1r1w_65536x72.json
[O]: reading $IFV_HOME/verilogparser/ip/1r1w_a127/ifv/properties.json
[O]: reading frun_cfg.json
[O]: cut=memoir_1r1w_65536x72 algo=1r1w_a127 noob=false version=3.3.6693
[I]: processing dout_check.log           constraints=15 assertions=02
proven=02 ---Test Passed!
[I]: processing fakemem_check.log        constraints=08 assertions=01
proven=01 ---Test Passed!
[I]: processing forward_check.log        constraints=09 assertions=05
proven=05 ---Test Passed!
[I]: processing mem_check.log            constraints=05 assertions=06
proven=06 ---Test Passed!
[I]: processing mem_range_check.log      constraints=06 assertions=04
proven=04 ---Test Passed!
[I]: processing np2_check.log            constraints=06 assertions=06
proven=06 ---Test Passed!
[I]: processing pdout_check.log          constraints=18 assertions=05
proven=05 ---Test Passed!
[I]: processing port_check.log           constraints=04 assertions=04
proven=04 ---Test Passed!
[I]: processing rmem_check.log           constraints=05 assertions=01
proven=01 ---Test Passed!
[I]: processing sold_fwd_check.log       constraints=06 assertions=02
proven=02 ---Test Passed!
[I]: processing srch_check.log           constraints=07 assertions=01
proven=01 ---Test Passed!
[I]: processing t1_1_mem_check.log       constraints=08 assertions=01
proven=01 ---Test Passed!
[I]: processing t1_1_port_check.log      constraints=04 assertions=01
proven=01 ---Test Passed!
[I]: processing t1_1_stack_check.log     constraints=03 assertions=08
proven=08 ---Test Passed!
[I]: processing t1_1_stack_dout_check.log constraints=04 assertions=04
proven=04 ---Test Passed!
[I]: processing t1_2_mem_check.log       constraints=08 assertions=01
proven=01 ---Test Passed!
[I]: processing t1_2_port_check.log      constraints=04 assertions=01
proven=01 ---Test Passed!
[I]: processing t1_2_stack_check.log     constraints=03 assertions=04
proven=04 ---Test Passed!
[I]: processing t1_2_stack_dout_check.log constraints=04 assertions=04
proven=04 ---Test Passed!
[I]: processing t1_align_check.log       constraints=04 assertions=02
proven=02 ---Test Passed!
[I]: processing t1_align_dout_check.log  constraints=05 assertions=03
proven=03 ---Test Passed!
[I]: processing t2d_align_check.log      constraints=06 assertions=04
proven=04 ---Test Passed!
```

```
[I]: processing t2d_align_dout_check.log constraints=08 assertions=06
proven=06 ---Test Passed!
[I]: processing t2d_mem_check.log        constraints=12 assertions=02
proven=02 ---Test Passed!
[I]: processing t2d_port_check.log       constraints=04 assertions=02
proven=02 ---Test Passed!
[I]: processing t2d_stack_check.log      constraints=06 assertions=04
proven=04 ---Test Passed!
[I]: processing t2d_stack_dout_check.log constraints=08 assertions=08
proven=08 ---Test Passed!
[I]: processing t2s_align_check.log      constraints=06 assertions=04
proven=04 ---Test Passed!
[I]: processing t2s_align_dout_check.log constraints=14 assertions=10
proven=10 ---Test Passed!
[I]: processing t2s_align_mem_check.log  constraints=06 assertions=02
proven=02 ---Test Passed!
[I]: processing t2s_mem_check.log        constraints=12 assertions=02
proven=02 ---Test Passed!
[I]: processing t2s_port_check.log       constraints=04 assertions=02
proven=02 ---Test Passed!
[I]: processing t2s_stack_check.log      constraints=06 assertions=04
proven=04 ---Test Passed!
[I]: processing t2s_stack_dout_check.log constraints=08 assertions=08
proven=08 ---Test Passed!
```

[O]: Total Assertions=124, Proven=124, Unproven=0