**Memory Interface**

module mem\_interface(

input wire clk,

input wire reset,

input wire req\_i,

input wire req\_rnw\_i,

input wire [3:0]req\_addr\_i,

input wire [31:0]req\_wdata\_i,

output wire req\_ready\_o,

output wire [31:0]req\_rdata\_o

);

//MEMORY ARRAY

`ifdef FORMAL

logic [31:0]mem[15:0];

`else

logic [15:0][31:0]mem;

`endif

logic mem\_rd;

logic mem\_wr;

logic req\_rising\_edge;

logic [3:0]lfsr\_val;

logic [3:0]count;

assign mem\_rd = req\_i & req\_rnw\_i;

assign mem\_wr = req\_i & ~req\_rnw\_i;

rising\_edge day3(

.clk(clk),

.reset(reset),

.a\_i(req\_i),

.rising\_edge\_o(req\_rising\_edge)

);

lfsr day7(

.clk(clk),

.reset(reset),

.lfsr\_o(lfsr\_val)

);

// Load a counter with random value on the rising edge

logic[3:0] count\_ff;

logic[3:0] nxt\_count;

always\_ff @(posedge clk or posedge reset)

if (reset)

count\_ff <= 4'h0;

else

count\_ff <= nxt\_count;

assign nxt\_count = req\_rising\_edge ? lfsr\_val:

count\_ff + 4'h1;

assign count = count\_ff;

// Write into the mem when the counter is 0

always\_ff @(posedge clk)

if (mem\_wr & ~|count)

mem[req\_addr\_i] <= req\_wdata\_i;

// Read directly

assign req\_rdata\_o = mem[req\_addr\_i] & {32{mem\_rd}};

// Assert ready only when counter is at 0

// This will add random delays on when memory gives the ready

assign req\_ready\_o = ~|count;

endmodule

**Test Bench Code**

module tb\_mem\_interface();

logic clk;

logic reset;

logic req\_i;

logic req\_rnw\_i;

logic[9:0] req\_addr\_i;

logic[31:0] req\_wdata\_i;

logic req\_ready\_o;

logic[31:0] req\_rdata\_o;

// Instatiate the RTL

mem\_interface DAY17 (.\*);

logic [9:0] [9:0] addr\_list;

// Generate the clock

always begin

clk = 1'b1;

#5;

clk = 1'b0;

#5;

end

// Generate stimulus

initial begin

reset <= 1'b1;

req\_i <= 1'b0;

@(posedge clk);

reset <= 1'b0;

@(posedge clk);

for (int txn=0; txn<10; txn++) begin

// Write 10 transactions

req\_i <= 1'b1;

req\_rnw\_i <= 0;

req\_addr\_i <= $urandom\_range(0, 1023);

addr\_list[txn] = req\_addr\_i;

req\_wdata\_i <= $urandom\_range(0, 32'hFFFF);

// Wait for ready

while (~req\_ready\_o) begin

@(posedge clk);

end

req\_i <= 1'b0;

@(posedge clk);

end

for (int txn=0; txn<10; txn++) begin

// Read 10 transactions

req\_i <= 1'b1;

req\_rnw\_i <= 1;

req\_addr\_i <= addr\_list[txn];

req\_wdata\_i <= $urandom\_range(0, 32'hFFFF);

// Wait for ready

while (~req\_ready\_o) begin

@(posedge clk);

end

req\_i <= 1'b0;

@(posedge clk);

end

$finish();

end

// Dump VCD

initial begin

$dumpfile("mem\_interface.vcd");

$dumpvars(0, tb\_mem\_interface);

end

**Simulation**

