module random\_stalls

(

input logic clk,

input logic core\_req\_i,

output logic core\_gnt\_o,

input logic [31:0] core\_addr\_i,

input logic core\_we\_i,

input logic [ 3:0] core\_be\_i,

input logic [31:0] core\_wdata\_i,

output logic [31:0] core\_rdata\_o,

output logic core\_rvalid\_o,

output logic data\_req\_o,

input logic data\_gnt\_i,

output logic [31:0] data\_addr\_o,

output logic data\_we\_o,

output logic [ 3:0] data\_be\_o,

output logic [31:0] data\_wdata\_o,

input logic [31:0] data\_rdata\_i,

input logic data\_rvalid\_i

);

class rand\_wait\_cycles;

rand int n;

constraint default\_c { n >= 0 ; n < 6;}

endclass

// random staller

typedef struct {

logic [31:0] addr;

logic we;

logic [ 3:0] be;

logic [31:0] wdata;

logic [31:0] rdata;

} stall\_mem\_t;

mailbox core\_reqs = new (4);

mailbox core\_resps = new (4);

mailbox core\_resps\_granted = new (4);

mailbox platform\_transfers = new (4);

// Core Request Side

// Waits for requests and puts them in a queue, does not perform actual

// requests to the platform

initial

begin

stall\_mem\_t mem\_acc;

automatic rand\_wait\_cycles wait\_cycles = new ();

int temp;

while(1) begin

core\_gnt\_o = 1'b0;

#1;

if (!core\_req\_i)

continue;

// we got a request, now let's wait for a random number of cycles before

// we give the grant

temp = wait\_cycles.randomize();

while(wait\_cycles.n != 0) begin

@(posedge clk);

wait\_cycles.n--;

#1;

end

// we waited for a random number of cycles, let's give the grant

core\_gnt\_o = 1'b1;

mem\_acc.addr = core\_addr\_i;

mem\_acc.be = core\_be\_i;

mem\_acc.we = core\_we\_i;

mem\_acc.wdata = core\_wdata\_i;

core\_reqs.put(mem\_acc);

@(posedge clk);

core\_resps\_granted.put(1'b1);

end

end

// Core Response Side

// Waits for a response from the platform and then waits for a random number

// of cycles before giving the rvalid

initial

begin

stall\_mem\_t mem\_acc;

automatic rand\_wait\_cycles wait\_cycles = new ();

logic granted;

int temp;

while(1) begin

@(posedge clk);

core\_rvalid\_o = 1'b0;

core\_rdata\_o = 'x;

core\_resps\_granted.get(granted);

core\_resps.get(mem\_acc);

// we got a response, now let's wait for a random amount of cycles

// we give the grant

temp = wait\_cycles.randomize();

while(wait\_cycles.n != 0) begin

@(posedge clk);

wait\_cycles.n--;

end

// we waited for a random number of cycles, let's give the rvalid

core\_rdata\_o = mem\_acc.rdata;

core\_rvalid\_o = 1'b1;

end

end

// platform request side

// Waits for requests from the core and then performs the request on the

// platform immediately

// Simulates a "virtual" core

initial

begin

stall\_mem\_t mem\_acc;

while(1) begin

@(posedge clk);

data\_req\_o = 1'b0;

data\_addr\_o = '0;

data\_we\_o = 1'b0;

data\_be\_o = 4'b0;

data\_wdata\_o = 'x;

core\_reqs.get(mem\_acc);

data\_req\_o = 1'b1;

data\_addr\_o = mem\_acc.addr;

data\_we\_o = mem\_acc.we;

data\_be\_o = mem\_acc.be;

data\_wdata\_o = mem\_acc.wdata;

#1;

while(!data\_gnt\_i) begin

@(posedge clk);

#1;

end

platform\_transfers.put(mem\_acc);

end

end

// platform response side

// Waits for rvalids and puts the responses into the core response mailbox

initial

begin

stall\_mem\_t mem\_acc;

while(1) begin

@(posedge clk);

platform\_transfers.get(mem\_acc);

while(!data\_rvalid\_i) begin

@(posedge clk);

end

mem\_acc.rdata = data\_rdata\_i;

core\_resps.put(mem\_acc);

end

end

endmodule