`include "axi\_bus.sv"

`include "config.sv"

module core\_region

#(

parameter AXI\_ADDR\_WIDTH = 32,

parameter AXI\_DATA\_WIDTH = 64,

parameter AXI\_ID\_MASTER\_WIDTH = 10,

parameter AXI\_ID\_SLAVE\_WIDTH = 10,

parameter AXI\_USER\_WIDTH = 0,

parameter DATA\_RAM\_SIZE = 32768, // in bytes

parameter INSTR\_RAM\_SIZE = 32768, // in bytes

parameter USE\_ZERO\_RISCY = 0,

parameter RISCY\_RV32F = 0,

parameter ZERO\_RV32M = 1,

parameter ZERO\_RV32E = 0

)

(

// Clock and Reset

input logic clk,

input logic rst\_n,

input logic testmode\_i,

input logic fetch\_enable\_i,

input logic [31:0] irq\_i,

output logic core\_busy\_o,

input logic clock\_gating\_i,

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

AXI\_BUS.Master core\_master,

AXI\_BUS.Master dbg\_master,

AXI\_BUS.Slave data\_slave,

AXI\_BUS.Slave instr\_slave,

DEBUG\_BUS.Slave debug,

// JTAG signals

input logic tck\_i,

input logic trstn\_i,

input logic tms\_i,

input logic tdi\_i,

output logic tdo\_o

);

localparam INSTR\_ADDR\_WIDTH = $clog2(INSTR\_RAM\_SIZE)+1; // to make space for the boot rom

localparam DATA\_ADDR\_WIDTH = $clog2(DATA\_RAM\_SIZE);

localparam AXI\_B\_WIDTH = $clog2(AXI\_DATA\_WIDTH/8); // AXI "Byte" width

// signals from/to core

logic core\_instr\_req;

logic core\_instr\_gnt;

logic core\_instr\_rvalid;

logic [31:0] core\_instr\_addr;

logic [31:0] core\_instr\_rdata;

logic core\_lsu\_req;

logic core\_lsu\_gnt;

logic core\_lsu\_rvalid;

logic [31:0] core\_lsu\_addr;

logic core\_lsu\_we;

logic [3:0] core\_lsu\_be;

logic [31:0] core\_lsu\_rdata;

logic [31:0] core\_lsu\_wdata;

logic core\_data\_req;

logic core\_data\_gnt;

logic core\_data\_rvalid;

logic [31:0] core\_data\_addr;

logic core\_data\_we;

logic [3:0] core\_data\_be;

logic [31:0] core\_data\_rdata;

logic [31:0] core\_data\_wdata;

// signals to/from AXI mem

logic is\_axi\_addr;

logic axi\_mem\_req;

logic [DATA\_ADDR\_WIDTH-1:0] axi\_mem\_addr;

logic axi\_mem\_we;

logic [AXI\_DATA\_WIDTH/8-1:0] axi\_mem\_be;

logic [AXI\_DATA\_WIDTH-1:0] axi\_mem\_rdata;

logic [AXI\_DATA\_WIDTH-1:0] axi\_mem\_wdata;

// signals to/from AXI instr

logic axi\_instr\_req;

logic [INSTR\_ADDR\_WIDTH-1:0] axi\_instr\_addr;

logic axi\_instr\_we;

logic [AXI\_DATA\_WIDTH/8-1:0] axi\_instr\_be;

logic [AXI\_DATA\_WIDTH-1:0] axi\_instr\_rdata;

logic [AXI\_DATA\_WIDTH-1:0] axi\_instr\_wdata;

// signals to/from instr mem

logic instr\_mem\_en;

logic [INSTR\_ADDR\_WIDTH-1:0] instr\_mem\_addr;

logic instr\_mem\_we;

logic [AXI\_DATA\_WIDTH/8-1:0] instr\_mem\_be;

logic [AXI\_DATA\_WIDTH-1:0] instr\_mem\_rdata;

logic [AXI\_DATA\_WIDTH-1:0] instr\_mem\_wdata;

// signals to/from data mem

logic data\_mem\_en;

logic [DATA\_ADDR\_WIDTH-1:0] data\_mem\_addr;

logic data\_mem\_we;

logic [AXI\_DATA\_WIDTH/8-1:0] data\_mem\_be;

logic [AXI\_DATA\_WIDTH-1:0] data\_mem\_rdata;

logic [AXI\_DATA\_WIDTH-1:0] data\_mem\_wdata;

enum logic [0:0] { AXI, RAM } lsu\_resp\_CS, lsu\_resp\_NS;

// signals to/from core2axi

logic core\_axi\_req;

logic core\_axi\_gnt;

logic core\_axi\_rvalid;

logic [31:0] core\_axi\_addr;

logic core\_axi\_we;

logic [3:0] core\_axi\_be;

logic [31:0] core\_axi\_rdata;

logic [31:0] core\_axi\_wdata;

AXI\_BUS

#(

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_MASTER\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH )

)

core\_master\_int();

//----------------------------------------------------------------------------//

// Core Instantiation

//----------------------------------------------------------------------------//

logic [4:0] irq\_id;

always\_comb begin

irq\_id = '0;

for (int i = 0; i < 32; i+=1) begin

if(irq\_i[i]) begin

irq\_id = i[4:0];

end

end

end

if(USE\_ZERO\_RISCY) begin: CORE

zeroriscy\_core

#(

.N\_EXT\_PERF\_COUNTERS ( 0 ),

.RV32E ( ZERO\_RV32E ),

.RV32M ( ZERO\_RV32M )

)

RISCV\_CORE

(

.clk\_i ( clk ),

.rst\_ni ( rst\_n ),

.clock\_en\_i ( clock\_gating\_i ),

.test\_en\_i ( testmode\_i ),

.boot\_addr\_i ( boot\_addr\_i ),

.core\_id\_i ( 4'h0 ),

.cluster\_id\_i ( 6'h0 ),

.instr\_addr\_o ( core\_instr\_addr ),

.instr\_req\_o ( core\_instr\_req ),

.instr\_rdata\_i ( core\_instr\_rdata ),

.instr\_gnt\_i ( core\_instr\_gnt ),

.instr\_rvalid\_i ( core\_instr\_rvalid ),

.data\_addr\_o ( core\_lsu\_addr ),

.data\_wdata\_o ( core\_lsu\_wdata ),

.data\_we\_o ( core\_lsu\_we ),

.data\_req\_o ( core\_lsu\_req ),

.data\_be\_o ( core\_lsu\_be ),

.data\_rdata\_i ( core\_lsu\_rdata ),

.data\_gnt\_i ( core\_lsu\_gnt ),

.data\_rvalid\_i ( core\_lsu\_rvalid ),

.data\_err\_i ( 1'b0 ),

.irq\_i ( (|irq\_i) ),

.irq\_id\_i ( irq\_id ),

.irq\_ack\_o ( ),

.irq\_id\_o ( ),

.debug\_req\_i ( debug.req ),

.debug\_gnt\_o ( debug.gnt ),

.debug\_rvalid\_o ( debug.rvalid ),

.debug\_addr\_i ( debug.addr ),

.debug\_we\_i ( debug.we ),

.debug\_wdata\_i ( debug.wdata ),

.debug\_rdata\_o ( debug.rdata ),

.debug\_halted\_o ( ),

.debug\_halt\_i ( 1'b0 ),

.debug\_resume\_i ( 1'b0 ),

.fetch\_enable\_i ( fetch\_enable\_i ),

.core\_busy\_o ( core\_busy\_o ),

.ext\_perf\_counters\_i ( )

);

end else begin: CORE

riscv\_core

#(

.N\_EXT\_PERF\_COUNTERS ( 0 ),

.FPU ( RISCY\_RV32F ),

.SHARED\_FP ( 0 ),

.SHARED\_FP\_DIVSQRT ( 2 )

)

RISCV\_CORE

(

.clk\_i ( clk ),

.rst\_ni ( rst\_n ),

.clock\_en\_i ( clock\_gating\_i ),

.test\_en\_i ( testmode\_i ),

.boot\_addr\_i ( boot\_addr\_i ),

.core\_id\_i ( 4'h0 ),

.cluster\_id\_i ( 6'h0 ),

.instr\_addr\_o ( core\_instr\_addr ),

.instr\_req\_o ( core\_instr\_req ),

.instr\_rdata\_i ( core\_instr\_rdata ),

.instr\_gnt\_i ( core\_instr\_gnt ),

.instr\_rvalid\_i ( core\_instr\_rvalid ),

.data\_addr\_o ( core\_lsu\_addr ),

.data\_wdata\_o ( core\_lsu\_wdata ),

.data\_we\_o ( core\_lsu\_we ),

.data\_req\_o ( core\_lsu\_req ),

.data\_be\_o ( core\_lsu\_be ),

.data\_rdata\_i ( core\_lsu\_rdata ),

.data\_gnt\_i ( core\_lsu\_gnt ),

.data\_rvalid\_i ( core\_lsu\_rvalid ),

.data\_err\_i ( 1'b0 ),

.irq\_i ( (|irq\_i) ),

.irq\_id\_i ( irq\_id ),

.irq\_ack\_o ( ),

.irq\_id\_o ( ),

.irq\_sec\_i ( 1'b0 ),

.sec\_lvl\_o ( ),

.debug\_req\_i ( debug.req ),

.debug\_gnt\_o ( debug.gnt ),

.debug\_rvalid\_o ( debug.rvalid ),

.debug\_addr\_i ( debug.addr ),

.debug\_we\_i ( debug.we ),

.debug\_wdata\_i ( debug.wdata ),

.debug\_rdata\_o ( debug.rdata ),

.debug\_halted\_o ( ),

.debug\_halt\_i ( 1'b0 ),

.debug\_resume\_i ( 1'b0 ),

.fetch\_enable\_i ( fetch\_enable\_i ),

.core\_busy\_o ( core\_busy\_o ),

// apu-interconnect

// handshake signals

.apu\_master\_req\_o ( ),

.apu\_master\_ready\_o ( ),

.apu\_master\_gnt\_i ( 1'b1 ),

// request channel

.apu\_master\_operands\_o ( ),

.apu\_master\_op\_o ( ),

.apu\_master\_type\_o ( ),

.apu\_master\_flags\_o ( ),

// response channel

.apu\_master\_valid\_i ( '0 ),

.apu\_master\_result\_i ( '0 ),

.apu\_master\_flags\_i ( '0 ),

.ext\_perf\_counters\_i ( )

);

end

core2axi\_wrap

#(

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_MASTER\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH ),

.REGISTERED\_GRANT ( "FALSE" )

)

core2axi\_i

(

.clk\_i ( clk ),

.rst\_ni ( rst\_n ),

.data\_req\_i ( core\_axi\_req ),

.data\_gnt\_o ( core\_axi\_gnt ),

.data\_rvalid\_o ( core\_axi\_rvalid ),

.data\_addr\_i ( core\_axi\_addr ),

.data\_we\_i ( core\_axi\_we ),

.data\_be\_i ( core\_axi\_be ),

.data\_rdata\_o ( core\_axi\_rdata ),

.data\_wdata\_i ( core\_axi\_wdata ),

.master ( core\_master\_int )

);

//----------------------------------------------------------------------------//

// AXI Slices

//----------------------------------------------------------------------------//

axi\_slice\_wrap

#(

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_MASTER\_WIDTH ),

.SLICE\_DEPTH ( 2 )

)

axi\_slice\_core2axi

(

.clk\_i ( clk ),

.rst\_ni ( rst\_n ),

.test\_en\_i ( testmode\_i ),

.axi\_slave ( core\_master\_int ),

.axi\_master ( core\_master )

);

//----------------------------------------------------------------------------//

// DEMUX

//----------------------------------------------------------------------------//

assign is\_axi\_addr = (core\_lsu\_addr[31:20] != 12'h001);

assign core\_data\_req = (~is\_axi\_addr) & core\_lsu\_req;

assign core\_axi\_req = is\_axi\_addr & core\_lsu\_req;

assign core\_data\_addr = core\_lsu\_addr;

assign core\_data\_we = core\_lsu\_we;

assign core\_data\_be = core\_lsu\_be;

assign core\_data\_wdata = core\_lsu\_wdata;

assign core\_axi\_addr = core\_lsu\_addr;

assign core\_axi\_we = core\_lsu\_we;

assign core\_axi\_be = core\_lsu\_be;

assign core\_axi\_wdata = core\_lsu\_wdata;

always\_ff @(posedge clk, negedge rst\_n)

begin

if (rst\_n == 1'b0)

lsu\_resp\_CS <= RAM;

else

lsu\_resp\_CS <= lsu\_resp\_NS;

end

// figure out where the next response will be coming from

always\_comb

begin

lsu\_resp\_NS = lsu\_resp\_CS;

core\_lsu\_gnt = 1'b0;

if (core\_axi\_req)

begin

core\_lsu\_gnt = core\_axi\_gnt;

lsu\_resp\_NS = AXI;

end

else if (core\_data\_req)

begin

core\_lsu\_gnt = core\_data\_gnt;

lsu\_resp\_NS = RAM;

end

end

// route response back to LSU

assign core\_lsu\_rdata = (lsu\_resp\_CS == AXI) ? core\_axi\_rdata : core\_data\_rdata;

assign core\_lsu\_rvalid = core\_axi\_rvalid | core\_data\_rvalid;

//----------------------------------------------------------------------------//

// Instruction RAM

//----------------------------------------------------------------------------//

instr\_ram\_wrap

#(

.RAM\_SIZE ( INSTR\_RAM\_SIZE ),

.DATA\_WIDTH ( AXI\_DATA\_WIDTH )

)

instr\_mem

(

.clk ( clk ),

.rst\_n ( rst\_n ),

.en\_i ( instr\_mem\_en ),

.addr\_i ( instr\_mem\_addr ),

.wdata\_i ( instr\_mem\_wdata ),

.rdata\_o ( instr\_mem\_rdata ),

.we\_i ( instr\_mem\_we ),

.be\_i ( instr\_mem\_be ),

.bypass\_en\_i ( testmode\_i )

);

axi\_mem\_if\_SP\_wrap

#(

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_SLAVE\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH ),

.MEM\_ADDR\_WIDTH ( INSTR\_ADDR\_WIDTH )

)

instr\_mem\_axi\_if

(

.clk ( clk ),

.rst\_n ( rst\_n ),

.test\_en\_i ( testmode\_i ),

.mem\_req\_o ( axi\_instr\_req ),

.mem\_addr\_o ( axi\_instr\_addr ),

.mem\_we\_o ( axi\_instr\_we ),

.mem\_be\_o ( axi\_instr\_be ),

.mem\_rdata\_i ( axi\_instr\_rdata ),

.mem\_wdata\_o ( axi\_instr\_wdata ),

.slave ( instr\_slave )

);

ram\_mux

#(

.ADDR\_WIDTH ( INSTR\_ADDR\_WIDTH ),

.IN0\_WIDTH ( AXI\_DATA\_WIDTH ),

.IN1\_WIDTH ( 32 ),

.OUT\_WIDTH ( AXI\_DATA\_WIDTH )

)

instr\_ram\_mux\_i

(

.clk ( clk ),

.rst\_n ( rst\_n ),

.port0\_req\_i ( axi\_instr\_req ),

.port0\_gnt\_o ( ),

.port0\_rvalid\_o ( ),

.port0\_addr\_i ( {axi\_instr\_addr[INSTR\_ADDR\_WIDTH-AXI\_B\_WIDTH-1:0], {AXI\_B\_WIDTH{1'b0}}} ),

.port0\_we\_i ( axi\_instr\_we ),

.port0\_be\_i ( axi\_instr\_be ),

.port0\_rdata\_o ( axi\_instr\_rdata ),

.port0\_wdata\_i ( axi\_instr\_wdata ),

.port1\_req\_i ( core\_instr\_req ),

.port1\_gnt\_o ( core\_instr\_gnt ),

.port1\_rvalid\_o ( core\_instr\_rvalid ),

.port1\_addr\_i ( core\_instr\_addr[INSTR\_ADDR\_WIDTH-1:0] ),

.port1\_we\_i ( 1'b0 ),

.port1\_be\_i ( '1 ),

.port1\_rdata\_o ( core\_instr\_rdata ),

.port1\_wdata\_i ( '0 ),

.ram\_en\_o ( instr\_mem\_en ),

.ram\_addr\_o ( instr\_mem\_addr ),

.ram\_we\_o ( instr\_mem\_we ),

.ram\_be\_o ( instr\_mem\_be ),

.ram\_rdata\_i ( instr\_mem\_rdata ),

.ram\_wdata\_o ( instr\_mem\_wdata )

);

//----------------------------------------------------------------------------//

// Data RAM

//----------------------------------------------------------------------------//

sp\_ram\_wrap

#(

.RAM\_SIZE ( DATA\_RAM\_SIZE ),

.DATA\_WIDTH ( AXI\_DATA\_WIDTH )

)

data\_mem

(

.clk ( clk ),

.rstn\_i ( rst\_n ),

.en\_i ( data\_mem\_en ),

.addr\_i ( data\_mem\_addr ),

.wdata\_i ( data\_mem\_wdata ),

.rdata\_o ( data\_mem\_rdata ),

.we\_i ( data\_mem\_we ),

.be\_i ( data\_mem\_be ),

.bypass\_en\_i ( testmode\_i )

);

axi\_mem\_if\_SP\_wrap

#(

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_SLAVE\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH ),

.MEM\_ADDR\_WIDTH ( DATA\_ADDR\_WIDTH )

)

data\_mem\_axi\_if

(

.clk ( clk ),

.rst\_n ( rst\_n ),

.test\_en\_i ( testmode\_i ),

.mem\_req\_o ( axi\_mem\_req ),

.mem\_addr\_o ( axi\_mem\_addr ),

.mem\_we\_o ( axi\_mem\_we ),

.mem\_be\_o ( axi\_mem\_be ),

.mem\_rdata\_i ( axi\_mem\_rdata ),

.mem\_wdata\_o ( axi\_mem\_wdata ),

.slave ( data\_slave )

);

ram\_mux

#(

.ADDR\_WIDTH ( DATA\_ADDR\_WIDTH ),

.IN0\_WIDTH ( AXI\_DATA\_WIDTH ),

.IN1\_WIDTH ( 32 ),

.OUT\_WIDTH ( AXI\_DATA\_WIDTH )

)

data\_ram\_mux\_i

(

.clk ( clk ),

.rst\_n ( rst\_n ),

.port0\_req\_i ( axi\_mem\_req ),

.port0\_gnt\_o ( ),

.port0\_rvalid\_o ( ),

.port0\_addr\_i ( {axi\_mem\_addr[DATA\_ADDR\_WIDTH-AXI\_B\_WIDTH-1:0], {AXI\_B\_WIDTH{1'b0}}} ),

.port0\_we\_i ( axi\_mem\_we ),

.port0\_be\_i ( axi\_mem\_be ),

.port0\_rdata\_o ( axi\_mem\_rdata ),

.port0\_wdata\_i ( axi\_mem\_wdata ),

.port1\_req\_i ( core\_data\_req ),

.port1\_gnt\_o ( core\_data\_gnt ),

.port1\_rvalid\_o ( core\_data\_rvalid ),

.port1\_addr\_i ( core\_data\_addr[DATA\_ADDR\_WIDTH-1:0] ),

.port1\_we\_i ( core\_data\_we ),

.port1\_be\_i ( core\_data\_be ),

.port1\_rdata\_o ( core\_data\_rdata ),

.port1\_wdata\_i ( core\_data\_wdata ),

.ram\_en\_o ( data\_mem\_en ),

.ram\_addr\_o ( data\_mem\_addr ),

.ram\_we\_o ( data\_mem\_we ),

.ram\_be\_o ( data\_mem\_be ),

.ram\_rdata\_i ( data\_mem\_rdata ),

.ram\_wdata\_o ( data\_mem\_wdata )

);

//----------------------------------------------------------------------------//

// Advanced Debug Unit

//----------------------------------------------------------------------------//

// TODO: remove the debug connections to the core

adv\_dbg\_if

#(

.NB\_CORES ( 1 ),

.AXI\_ADDR\_WIDTH ( AXI\_ADDR\_WIDTH ),

.AXI\_DATA\_WIDTH ( AXI\_DATA\_WIDTH ),

.AXI\_USER\_WIDTH ( AXI\_USER\_WIDTH ),

.AXI\_ID\_WIDTH ( AXI\_ID\_MASTER\_WIDTH )

)

adv\_dbg\_if\_i

(

.tms\_pad\_i ( tms\_i ),

.tck\_pad\_i ( tck\_i ),

.trstn\_pad\_i ( trstn\_i ),

.tdi\_pad\_i ( tdi\_i ),

.tdo\_pad\_o ( tdo\_o ),

.test\_mode\_i ( testmode\_i ),

.cpu\_addr\_o ( ),

.cpu\_data\_i ( '0 ),

.cpu\_data\_o ( ),

.cpu\_bp\_i ( '0 ),

.cpu\_stall\_o ( ),

.cpu\_stb\_o ( ),

.cpu\_we\_o ( ),

.cpu\_ack\_i ( '1 ),

.cpu\_rst\_o ( ),

.axi\_aclk ( clk ),

.axi\_aresetn ( rst\_n ),

.axi\_master\_aw\_valid ( dbg\_master.aw\_valid ),

.axi\_master\_aw\_addr ( dbg\_master.aw\_addr ),

.axi\_master\_aw\_prot ( dbg\_master.aw\_prot ),

.axi\_master\_aw\_region ( dbg\_master.aw\_region ),

.axi\_master\_aw\_len ( dbg\_master.aw\_len ),

.axi\_master\_aw\_size ( dbg\_master.aw\_size ),

.axi\_master\_aw\_burst ( dbg\_master.aw\_burst ),

.axi\_master\_aw\_lock ( dbg\_master.aw\_lock ),

.axi\_master\_aw\_cache ( dbg\_master.aw\_cache ),

.axi\_master\_aw\_qos ( dbg\_master.aw\_qos ),

.axi\_master\_aw\_id ( dbg\_master.aw\_id ),

.axi\_master\_aw\_user ( dbg\_master.aw\_user ),

.axi\_master\_aw\_ready ( dbg\_master.aw\_ready ),

.axi\_master\_ar\_valid ( dbg\_master.ar\_valid ),

.axi\_master\_ar\_addr ( dbg\_master.ar\_addr ),

.axi\_master\_ar\_prot ( dbg\_master.ar\_prot ),

.axi\_master\_ar\_region ( dbg\_master.ar\_region ),

.axi\_master\_ar\_len ( dbg\_master.ar\_len ),

.axi\_master\_ar\_size ( dbg\_master.ar\_size ),

.axi\_master\_ar\_burst ( dbg\_master.ar\_burst ),

.axi\_master\_ar\_lock ( dbg\_master.ar\_lock ),

.axi\_master\_ar\_cache ( dbg\_master.ar\_cache ),

.axi\_master\_ar\_qos ( dbg\_master.ar\_qos ),

.axi\_master\_ar\_id ( dbg\_master.ar\_id ),

.axi\_master\_ar\_user ( dbg\_master.ar\_user ),

.axi\_master\_ar\_ready ( dbg\_master.ar\_ready ),

.axi\_master\_w\_valid ( dbg\_master.w\_valid ),

.axi\_master\_w\_data ( dbg\_master.w\_data ),

.axi\_master\_w\_strb ( dbg\_master.w\_strb ),

.axi\_master\_w\_user ( dbg\_master.w\_user ),

.axi\_master\_w\_last ( dbg\_master.w\_last ),

.axi\_master\_w\_ready ( dbg\_master.w\_ready ),

.axi\_master\_r\_valid ( dbg\_master.r\_valid ),

.axi\_master\_r\_data ( dbg\_master.r\_data ),

.axi\_master\_r\_resp ( dbg\_master.r\_resp ),

.axi\_master\_r\_last ( dbg\_master.r\_last ),

.axi\_master\_r\_id ( dbg\_master.r\_id ),

.axi\_master\_r\_user ( dbg\_master.r\_user ),

.axi\_master\_r\_ready ( dbg\_master.r\_ready ),

.axi\_master\_b\_valid ( dbg\_master.b\_valid ),

.axi\_master\_b\_resp ( dbg\_master.b\_resp ),

.axi\_master\_b\_id ( dbg\_master.b\_id ),

.axi\_master\_b\_user ( dbg\_master.b\_user ),

.axi\_master\_b\_ready ( dbg\_master.b\_ready )

);

//----------------------------------------------------------------------------//

// Test Code

//----------------------------------------------------------------------------//

// introduce random stalls for data access to stress LSU

`ifdef DATA\_STALL\_RANDOM

random\_stalls data\_stalls\_i

(

.clk ( clk ),

.core\_req\_i ( CORE.RISCV\_CORE.data\_req\_o ),

.core\_addr\_i ( CORE.RISCV\_CORE.data\_addr\_o ),

.core\_we\_i ( CORE.RISCV\_CORE.data\_we\_o ),

.core\_be\_i ( CORE.RISCV\_CORE.data\_be\_o ),

.core\_wdata\_i ( CORE.RISCV\_CORE.data\_wdata\_o ),

.core\_gnt\_o ( ),

.core\_rdata\_o ( ),

.core\_rvalid\_o ( ),

.data\_req\_o ( ),

.data\_addr\_o ( ),

.data\_we\_o ( ),

.data\_be\_o ( ),

.data\_wdata\_o ( ),

.data\_gnt\_i ( core\_lsu\_gnt ),

.data\_rdata\_i ( core\_lsu\_rdata ),

.data\_rvalid\_i ( core\_lsu\_rvalid )

);

initial begin

force CORE.RISCV\_CORE.data\_gnt\_i = data\_stalls\_i.core\_gnt\_o;

force CORE.RISCV\_CORE.data\_rvalid\_i = data\_stalls\_i.core\_rvalid\_o;

force CORE.RISCV\_CORE.data\_rdata\_i = data\_stalls\_i.core\_rdata\_o;

force core\_lsu\_req = data\_stalls\_i.data\_req\_o;

force core\_lsu\_addr = data\_stalls\_i.data\_addr\_o;

force core\_lsu\_we = data\_stalls\_i.data\_we\_o;

force core\_lsu\_be = data\_stalls\_i.data\_be\_o;

force core\_lsu\_wdata = data\_stalls\_i.data\_wdata\_o;

end

`endif

// introduce random stalls for instruction access to stress instruction

// fetcher

`ifdef INSTR\_STALL\_RANDOM

random\_stalls instr\_stalls\_i

(

.clk ( clk ),

.core\_req\_i ( CORE.RISCV\_CORE.instr\_req\_o ),

.core\_addr\_i ( CORE.RISCV\_CORE.instr\_addr\_o ),

.core\_we\_i ( ),

.core\_be\_i ( ),

.core\_wdata\_i ( ),

.core\_gnt\_o ( ),

.core\_rdata\_o ( ),

.core\_rvalid\_o ( ),

.data\_req\_o ( ),

.data\_addr\_o ( ),

.data\_we\_o ( ),

.data\_be\_o ( ),

.data\_wdata\_o ( ),

.data\_gnt\_i ( core\_instr\_gnt ),

.data\_rdata\_i ( core\_instr\_rdata ),

.data\_rvalid\_i ( core\_instr\_rvalid )

);

initial begin

force CORE.RISCV\_CORE.instr\_gnt\_i = instr\_stalls\_i.core\_gnt\_o;

force CORE.RISCV\_CORE.instr\_rvalid\_i = instr\_stalls\_i.core\_rvalid\_o;

force CORE.RISCV\_CORE.instr\_rdata\_i = instr\_stalls\_i.core\_rdata\_o;

force core\_instr\_req = instr\_stalls\_i.data\_req\_o;

force core\_instr\_addr = instr\_stalls\_i.data\_addr\_o;

end

`endif

endmodule