module ram\_mux

#(

parameter ADDR\_WIDTH = 32,

parameter OUT\_WIDTH = 32,

parameter IN0\_WIDTH = 32, // in widths cannot be bigger than out width

parameter IN1\_WIDTH = 32

)(

// Clock and Reset

input logic clk,

input logic rst\_n,

// port0 has priority

input logic port0\_req\_i,

output logic port0\_gnt\_o,

output logic port0\_rvalid\_o,

input logic [ADDR\_WIDTH-1:0] port0\_addr\_i,

input logic port0\_we\_i,

input logic [IN0\_WIDTH/8-1:0] port0\_be\_i,

output logic [IN0\_WIDTH-1:0] port0\_rdata\_o,

input logic [IN0\_WIDTH-1:0] port0\_wdata\_i,

input logic port1\_req\_i,

output logic port1\_gnt\_o,

output logic port1\_rvalid\_o,

input logic [ADDR\_WIDTH-1:0] port1\_addr\_i,

input logic port1\_we\_i,

input logic [IN1\_WIDTH/8-1:0] port1\_be\_i,

output logic [IN1\_WIDTH-1:0] port1\_rdata\_o,

input logic [IN1\_WIDTH-1:0] port1\_wdata\_i,

// to RAM

output logic ram\_en\_o,

output logic [ADDR\_WIDTH-1:0] ram\_addr\_o,

output logic ram\_we\_o,

output logic [OUT\_WIDTH/8-1:0] ram\_be\_o,

input logic [OUT\_WIDTH-1:0] ram\_rdata\_i,

output logic [OUT\_WIDTH-1:0] ram\_wdata\_o

);

localparam IN0\_ADDR\_HIGH = $clog2(OUT\_WIDTH/8) - 1;

localparam IN0\_ADDR\_LOW = $clog2(IN0\_WIDTH/8);

localparam IN0\_RATIO = OUT\_WIDTH/IN0\_WIDTH;

logic [OUT\_WIDTH/8-1:0] port0\_be;

// size adaptation

genvar i0;

generate

if (IN0\_ADDR\_HIGH >= IN0\_ADDR\_LOW)

begin

logic port0\_addr\_q;

logic [IN0\_RATIO-1:0][IN0\_WIDTH-1:0] port0\_rdata;

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

begin

if (~rst\_n)

port0\_addr\_q <= '0;

else

if (port0\_gnt\_o)

port0\_addr\_q <= port0\_addr\_i[IN0\_ADDR\_HIGH:IN0\_ADDR\_LOW];

end

for(i0 = 0; i0 < IN0\_RATIO; i0++)

begin

assign port0\_be[(i0+1) \* IN0\_WIDTH/8 -1:i0\*IN0\_WIDTH/8] =

(i0 == port0\_addr\_i[IN0\_ADDR\_HIGH:IN0\_ADDR\_LOW]) ? port0\_be\_i : '0;

assign port0\_rdata[i0] = ram\_rdata\_i[(i0+1)\*IN0\_WIDTH-1:i0\*IN0\_WIDTH];

end

assign port0\_rdata\_o = port0\_rdata[port0\_addr\_q];

end else begin

// nothing to be done, just pass through

assign port0\_be = port0\_be\_i;

assign port0\_rdata\_o = ram\_rdata\_i;

end

endgenerate

localparam IN1\_ADDR\_HIGH = $clog2(OUT\_WIDTH/8) - 1;

localparam IN1\_ADDR\_LOW = $clog2(IN1\_WIDTH/8);

localparam IN1\_RATIO = OUT\_WIDTH/IN1\_WIDTH;

logic [OUT\_WIDTH/8-1:0] port1\_be;

// size adaptation

genvar i1;

generate

if (IN1\_ADDR\_HIGH >= IN1\_ADDR\_LOW)

begin

logic port1\_addr\_q;

logic [IN1\_RATIO-1:0][IN1\_WIDTH-1:0] port1\_rdata;

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

begin

if (~rst\_n)

port1\_addr\_q <= '0;

else

if (port1\_gnt\_o)

port1\_addr\_q <= port1\_addr\_i[IN1\_ADDR\_HIGH:IN1\_ADDR\_LOW];

end

for(i1 = 0; i1 < OUT\_WIDTH/IN1\_WIDTH; i1++)

begin

assign port1\_be[(i1+1) \* IN1\_WIDTH/8 -1:i1\*IN1\_WIDTH/8] =

(i1 == port1\_addr\_i[IN1\_ADDR\_HIGH:IN1\_ADDR\_LOW]) ? port1\_be\_i : '0;

assign port1\_rdata[i1] = ram\_rdata\_i[(i1+1)\*IN1\_WIDTH-1:i1\*IN1\_WIDTH];

end

assign port1\_rdata\_o = port1\_rdata[port1\_addr\_q];

end else begin

// nothing to be done, just pass through

assign port1\_be = port1\_be\_i;

assign port1\_rdata\_o = ram\_rdata\_i;

end

endgenerate

// RAM mux for data port of core and AXI interface

always\_comb

begin

port0\_gnt\_o = 1'b0;

port1\_gnt\_o = 1'b0;

// port0 has priority

if(port0\_req\_i)

port0\_gnt\_o = 1'b1;

else if(port1\_req\_i)

port1\_gnt\_o = 1'b1;

end

assign ram\_en\_o = port0\_req\_i | port1\_req\_i;

assign ram\_addr\_o = port0\_req\_i ? port0\_addr\_i : port1\_addr\_i;

assign ram\_wdata\_o = port0\_req\_i ? {OUT\_WIDTH/IN0\_WIDTH{port0\_wdata\_i}} : {OUT\_WIDTH/IN1\_WIDTH{port1\_wdata\_i}};

assign ram\_we\_o = port0\_req\_i ? port0\_we\_i : port1\_we\_i;

assign ram\_be\_o = port0\_req\_i ? port0\_be : port1\_be;

// generate rvalid signals from gnt signals

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

begin

if (rst\_n == 1'b0)

begin

port0\_rvalid\_o <= 1'b0;

port1\_rvalid\_o <= 1'b0;

end

else

begin

port0\_rvalid\_o <= port0\_gnt\_o;

port1\_rvalid\_o <= port1\_gnt\_o;

end

end

endmodule