module ahb\_top(

input hclk,

input hresetn,

input enable,

input [31:0] dina,

input [31:0] dinb,

input [31:0] addr,

input wr,

input [1:0] slave\_sel,

output [31:0] dout

);

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

// Connect wires

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

// master

wire [1:0] sel;

wire [31:0] haddr;

wire hwrite;

wire [3:0] hprot;

wire [2:0] hsize;

wire [2:0] hburst;

wire [1:0] htrans;

wire hmastlock;

wire hready;

wire [31:0] hwdata;

// slave 1

wire [31:0] hrdata\_1;

wire hreadyout\_1;

wire hresp\_1;

// slave 2

wire [31:0] hrdata\_2;

wire hreadyout\_2;

wire hresp\_2;

// slave 3

wire [31:0] hrdata\_3;

wire hreadyout\_3;

wire hresp\_3;

// slave 4

wire [31:0] hrdata\_4;

wire hreadyout\_4;

wire hresp\_4;

// decoder

wire hsel\_1;

wire hsel\_2;

wire hsel\_3;

wire hsel\_4;

// multiplexor

wire [31:0] hrdata;

wire hreadyout;

wire hresp;

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

// Connect master, slaves, decoder, multiplexor

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

// master

ahb\_master master(

.hclk(hclk),

.hresetn(hresetn),

.enable(enable),

.dina(dina),

.dinb(dinb),

.addr(addr),

.wr(wr),

.hreadyout(hreadyout),

.hresp(hresp),

.hrdata(hrdata),

.slave\_sel(slave\_sel),

.sel(sel),

.haddr(haddr),

.hsize(hsize),

.hwrite(hwrite),

.hburst(hburst),

.hprot(hprot),

.htrans(htrans),

.hmastlock(hmastlock),

.hready(hready),

.hwdata(hwdata),

.dout(dout)

);

// decoder

decoder deco(

.sel(sel),

.hsel\_1(hsel\_1),

.hsel\_2(hsel\_2),

.hsel\_3(hsel\_3),

.hsel\_4(hsel\_4)

);

// slave 1

ahb\_slave slave1(

.hclk(hclk),

.hresetn(hresetn),

.hsel(hsel\_1),

.haddr(haddr),

.hwrite(hwrite),

.hsize(hsize),

.hburst(hburst),

.hprot(hprot),

.htrans(htrans),

.hmastlock(hmastlock),

.hready(hready),

.hwdata(hwdata),

.hreadyout(hreadyout\_1),

.hresp(hresp\_1),

.hrdata(hrdata\_1)

);

// slave 2

ahb\_slave slave2(

.hclk(hclk),

.hresetn(hresetn),

.hsel(hsel\_2),

.haddr(haddr),

.hwrite(hwrite),

.hsize(hsize),

.hburst(hburst),

.hprot(hprot),

.htrans(htrans),

.hmastlock(hmastlock),

.hready(hready),

.hwdata(hwdata),

.hreadyout(hreadyout\_2),

.hresp(hresp\_2),

.hrdata(hrdata\_2)

);

// slave 3

ahb\_slave slave3(

.hclk(hclk),

.hresetn(hresetn),

.hsel(hsel\_3),

.haddr(haddr),

.hwrite(hwrite),

.hsize(hsize),

.hburst(hburst),

.hprot(hprot),

.htrans(htrans),

.hmastlock(hmastlock),

.hready(hready),

.hwdata(hwdata),

.hreadyout(hreadyout\_3),

.hresp(hresp\_3),

.hrdata(hrdata\_3)

);

// slave 4

ahb\_slave slave4(

.hclk(hclk),

.hresetn(hresetn),

.hsel(hsel\_4),

.haddr(haddr),

.hwrite(hwrite),

.hsize(hsize),

.hburst(hburst),

.hprot(hprot),

.htrans(htrans),

.hmastlock(hmastlock),

.hready(hready),

.hwdata(hwdata),

.hreadyout(hreadyout\_4),

.hresp(hresp\_4),

.hrdata(hrdata\_4)

);

// multiplexor

multiplexor multip(

.hrdata\_1(hrdata\_1),

.hrdata\_2(hrdata\_2),

.hrdata\_3(hrdata\_3),

.hrdata\_4(hrdata\_4),

.hreadyout\_1(hreadyout\_1),

.hreadyout\_2(hreadyout\_2),

.hreadyout\_3(hreadyout\_3),

.hreadyout\_4(hreadyout\_4),

.hresp\_1(hresp\_1),

.hresp\_2(hresp\_2),

.hresp\_3(hresp\_3),

.hresp\_4(hresp\_4),

.sel(sel),

.hrdata(hrdata),

.hreadyout(hreadyout),

.hresp(hresp)

);

endmodule

module ahb\_master(

input hclk,

input hresetn,

input enable,

input [31:0] dina,

input [31:0] dinb,

input [31:0] addr,

input wr,

input hreadyout,

input hresp,

input [31:0] hrdata,

input [1:0] slave\_sel,

output reg [1:0] sel,

output reg [31:0] haddr,

output reg hwrite,

output reg [2:0] hsize,

output reg [2:0] hburst,

output reg [3:0] hprot,

output reg [1:0] htrans,

output reg hmastlock,

output reg hready,

output reg [31:0] hwdata,

output reg [31:0] dout

);

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

// The definitions for state machine

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

reg [1:0] state, next\_state;

parameter idle = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;

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

// The state machine

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

always @(posedge hclk, negedge hresetn) begin

if(!hresetn) begin

state <= idle;

end

else begin

state <= next\_state;

end

end

always @(\*) begin

case(state)

idle: begin

if(enable == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

s1: begin

if(wr == 1'b1) begin

next\_state = s2;

end

else begin

next\_state = s3;

end

end

s2: begin

if(enable == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

s3: begin

if(enable == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

default: begin

next\_state = idle;

end

endcase

end

always @(posedge hclk, negedge hresetn) begin

if(!hresetn) begin

sel <= 2'b00;

haddr <= 32'h0000\_0000;

hwrite <= 1'b0;

hsize <= 3'b000;

hburst <= 3'b000;

hprot <= 4'b0000;

htrans <= 2'b00;

hmastlock <= 1'b0;

hready <= 1'b0;

hwdata <= 32'h0000\_0000;

dout <= 32'h0000\_0000;

end

else begin

case(next\_state)

idle: begin

sel <= slave\_sel;

haddr <= addr;

hwrite <= hwrite;

hburst <= hburst;

hready <= 1'b0;

hwdata <= hwdata;

dout <= dout;

end

s1: begin

sel <= slave\_sel;

haddr <= addr;

hwrite <= wr;

hburst <= 3'b000;

hready <= 1'b1;

hwdata <= dina+dinb;

dout <= dout;

end

s2: begin

sel <= sel;

haddr <= addr;

hwrite <= wr;

hburst <= 3'b000;

hready <= 1'b1;

hwdata <= dina+dinb;

dout <= dout;

end

s3: begin

sel <= sel;

haddr <= addr;

hwrite <= wr;

hburst <= 3'b000;

hready <= 1'b1;

hwdata <= hwdata;

dout <= hrdata;

end

default: begin

sel <= slave\_sel;

haddr <= haddr;

hwrite <= hwrite;

hburst <= hburst;

hready <= 1'b0;

hwdata <= hwdata;

dout <= dout;

end

endcase

end

end

endmodule

module ahb\_slave(

input hclk,

input hresetn,

input hsel,

input [31:0] haddr,

input hwrite,

input [2:0] hsize,

input [2:0] hburst,

input [3:0] hprot,

input [1:0] htrans,

input hmastlock,

input hready,

input [31:0] hwdata,

output reg hreadyout,

output reg hresp,

output reg [31:0] hrdata

);

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

// The definitions for internal registers for data storge

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

reg [31:0] mem [31:0];

reg [4:0] waddr;

reg [4:0] raddr;

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

// The definition for state machine

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

reg [1:0] state;

reg [1:0] next\_state;

localparam idle = 2'b00,s1 = 2'b01,s2 = 2'b10,s3 = 2'b11;

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

// The definition for burst feature

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

reg single\_flag;

reg incr\_flag;

reg wrap4\_flag;

reg incr4\_flag;

reg wrap8\_flag;

reg incr8\_flag;

reg wrap16\_flag;

reg incr16\_flag;

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

// The state machine

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

always @(posedge hclk, negedge hresetn) begin

if(!hresetn) begin

state <= idle;

end

else begin

state <= next\_state;

end

end

always @(\*) begin

case(state)

idle: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

if(hsel == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

s1: begin

case(hburst)

// single transfer burst

3'b000: begin

single\_flag = 1'b1;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// incrementing burst of undefined length

3'b001: begin

single\_flag = 1'b0;

incr\_flag = 1'b1;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// 4-beat wrapping burst

3'b010: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b1;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// 4-beat incrementing burst

3'b011: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b1;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// 8-beat wrapping burst

3'b100: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b1;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// 8-beat incrementing burst

3'b101: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b1;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

// 16-beat wrapping burst

3'b110: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b1;

incr16\_flag = 1'b0;

end

// 16-beat incrementing burst

3'b111: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b1;

end

// default

default: begin

single\_flag = 1'b0;

incr\_flag = 1'b0;

wrap4\_flag = 1'b0;

incr4\_flag = 1'b0;

wrap8\_flag = 1'b0;

incr8\_flag = 1'b0;

wrap16\_flag = 1'b0;

incr16\_flag = 1'b0;

end

endcase

if((hwrite == 1'b1) && (hready == 1'b1)) begin

next\_state = s2;

end

else if((hwrite == 1'b0) && (hready == 1'b1)) begin

next\_state = s3;

end

else begin

next\_state = s1;

end

end

s2: begin

case(hburst)

// single transfer burst

3'b000: begin

if(hsel == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

// incrementing burst of undefined length

3'b001: begin

next\_state = s2;

end

// 4-beat wrapping burst

3'b010: begin

next\_state = s2;

end

// 4-beat incrementing burst

3'b011: begin

next\_state = s2;

end

// 8-beat wrapping burst

3'b100: begin

next\_state = s2;

end

// 8-beat incrementing burst

3'b101: begin

next\_state = s2;

end

// 16-beat wrapping burst

3'b110: begin

next\_state = s2;

end

// 16-beat incrementing burst

3'b111: begin

next\_state = s2;

end

// default

default: begin

if(hsel == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

endcase

end

s3: begin

case(hburst)

// single transfer burst

3'b000: begin

if(hsel == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

// incrementing burst of undefined length

3'b001: begin

next\_state = s3;

end

// 4-beat wrapping burst

3'b010: begin

next\_state = s3;

end

// 4-beat incrementing burst

3'b011: begin

next\_state = s3;

end

// 8-beat wrapping burst

3'b100: begin

next\_state = s3;

end

// 8-beat incrementing burst

3'b101: begin

next\_state = s3;

end

// 16-beat wrapping burst

3'b110: begin

next\_state = s3;

end

// 16-beat incrementing burst

3'b111: begin

next\_state = s3;

end

// default

default: begin

if(hsel == 1'b1) begin

next\_state = s1;

end

else begin

next\_state = idle;

end

end

endcase

end

default: begin

next\_state = idle;

end

endcase

end

always @(posedge hclk, negedge hresetn) begin

if(!hresetn) begin

hreadyout <= 1'b0;

hresp <= 1'b0;

hrdata <= 32'h0000\_0000;

waddr <= 5'b0000\_0;

raddr <= 5'b0000\_0;

end

else begin

case(next\_state)

idle: begin

hreadyout <= 1'b0;

hresp <= 1'b0;

hrdata <= hrdata;

waddr <= waddr;

raddr <= raddr;

end

s1: begin

hreadyout <= 1'b0;

hresp <= 1'b0;

hrdata <= hrdata;

waddr <= haddr;

raddr <= haddr;

end

s2: begin

case({single\_flag,incr\_flag,wrap4\_flag,incr4\_flag,wrap8\_flag,incr8\_flag,wrap16\_flag,incr16\_flag})

// single transfer

8'b1000\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

mem[waddr] <= hwdata;

end

// incre

8'b0100\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

// wrap 4

8'b0010\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(waddr < (haddr + 2'd3)) begin

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

else begin

mem[waddr] <= hwdata;

waddr <= haddr;

end

end

// incre 4

8'b0001\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

// wrap 8

8'b0000\_1000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(waddr < (haddr + 3'd7)) begin

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

else begin

mem[waddr] <= hwdata;

waddr <= haddr;

end

end

// incre 8

8'b0000\_0100: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

// wrap 16

8'b0000\_0010: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(waddr < (haddr + 4'd15)) begin

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

else begin

mem[waddr] <= hwdata;

waddr <= haddr;

end

end

// incre 16

8'b0000\_0001: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

mem[waddr] <= hwdata;

waddr <= waddr + 1'b1;

end

// default

default: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

end

endcase

end

s3: begin

case({single\_flag,incr\_flag,wrap4\_flag,incr4\_flag,wrap8\_flag,incr8\_flag,wrap16\_flag,incr16\_flag})

// single transfer

8'b1000\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

hrdata <= mem[raddr];

end

// incre

8'b0100\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

// wrap 4

8'b0010\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(raddr < (haddr + 2'd3)) begin

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

else begin

hrdata <= mem[raddr];

raddr <= haddr;

end

end

// incre 4

8'b0001\_0000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

// wrap 8

8'b0000\_1000: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(raddr < (haddr + 3'd7)) begin

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

else begin

hrdata <= mem[raddr];

raddr <= haddr;

end

end

// incre 8

8'b0000\_0100: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

// wrap 16

8'b0000\_0010: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

if(raddr < (haddr + 4'd15)) begin

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

else begin

hrdata <= mem[raddr];

raddr <= haddr;

end

end

// incre 16

8'b0000\_0001: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

hrdata <= mem[raddr];

raddr <= raddr + 1'b1;

end

// default

default: begin

hreadyout <= 1'b1;

hresp <= 1'b0;

end

endcase

end

default: begin

hreadyout <= 1'b0;

hresp <= 1'b0;

hrdata <= hrdata;

waddr <= waddr;

raddr <= raddr;

end

endcase

end

end

endmodule

module multiplexor(

input [31:0] hrdata\_1,

input [31:0] hrdata\_2,

input [31:0] hrdata\_3,

input [31:0] hrdata\_4,

input hreadyout\_1,

input hreadyout\_2,

input hreadyout\_3,

input hreadyout\_4,

input hresp\_1,

input hresp\_2,

input hresp\_3,

input hresp\_4,

input [1:0] sel,

output reg [31:0] hrdata,

output reg hreadyout,

output reg hresp

);

always @(\*) begin

case(sel)

2'b00: begin

hrdata = hrdata\_1;

hreadyout = hreadyout\_1;

hresp = hresp\_1;

end

2'b01: begin

hrdata = hrdata\_2;

hreadyout = hreadyout\_2;

hresp = hresp\_2;

end

2'b10: begin

hrdata = hrdata\_3;

hreadyout = hreadyout\_3;

hresp = hresp\_3;

end

2'b11: begin

hrdata = hrdata\_4;

hreadyout = hreadyout\_4;

hresp = hresp\_4;

end

default: begin

hrdata = 32'h0000\_0000;

hreadyout = 1'b0;

hresp = 1'b0;

end

endcase

end

endmodule

module decoder(

input [1:0] sel,

output reg hsel\_1,

output reg hsel\_2,

output reg hsel\_3,

output reg hsel\_4

);

always @(\*) begin

case(sel)

2'b00: begin

hsel\_1 = 1'b1;

hsel\_2 = 1'b0;

hsel\_3 = 1'b0;

hsel\_4 = 1'b0;

end

2'b01: begin

hsel\_1 = 1'b0;

hsel\_2 = 1'b1;

hsel\_3 = 1'b0;

hsel\_4 = 1'b0;

end

2'b10: begin

hsel\_1 = 1'b0;

hsel\_2 = 1'b0;

hsel\_3 = 1'b1;

hsel\_4 = 1'b0;

end

2'b11: begin

hsel\_1 = 1'b0;

hsel\_2 = 1'b0;

hsel\_3 = 1'b0;

hsel\_4 = 1'b1;

end

default: begin

hsel\_1 = 1'b0;

hsel\_2 = 1'b0;

hsel\_3 = 1'b0;

hsel\_4 = 1'b0;

end

endcase

end

endmodule