`timescale 1ns/1ps

module memory1(

input clk,cs,wen,rst,

input [7:0] addr,

input [31:0] wr\_data,

output reg [31:0] rd\_data);

reg [31:0] mem[0:(1<<8)-1];

always @(posedge clk)

begin: MEM\_WRITE

if(cs & wen)

mem[addr] <= wr\_data;

end

always @(posedge clk)

begin: MEM\_READ

if(rst)

rd\_data <= 0;

else

begin

if(cs & (~wen))

rd\_data <= mem[addr];

end

end

endmodule

module reg\_rd\_wr(

input clk,rst,soc\_sel,soc\_wen,

input [19:0] soc\_addr,

input [31:0] soc\_wr\_data,

output reg [31:0] soc\_rd\_data);

reg [31:0] reg1; //mem\_sel

reg [31:0] reg2; //mem\_wen

reg [31:0] reg3; //mem\_addr

reg [31:0] reg4; //mem\_wr\_data

reg [31:0] reg5; //mem\_rd\_data

parameter ARRD\_REG1 = 20'd0, ARRD\_REG2 = 20'd4, ARRD\_REG3 = 20'd8, ARRD\_REG4 = 20'd12, ARRD\_REG5 = 20'd16;

wire [31:0] w\_rd\_data\_out;

always@(posedge clk)

begin

if(rst)

begin

reg1 <= 0;

reg2 <= 0;

reg3 <= 0;

reg4 <= 0;

reg5 <= 0;

end

else

begin

if(soc\_sel & soc\_wen)

begin

case(soc\_addr)

ARRD\_REG1: reg1 <= soc\_wr\_data;

ARRD\_REG2: reg2 <= soc\_wr\_data;

ARRD\_REG3: reg3 <= soc\_wr\_data;

ARRD\_REG4: reg4 <= soc\_wr\_data;

ARRD\_REG5: reg5 <= soc\_wr\_data;

default: begin reg1 <= soc\_wr\_data; end

endcase

end

end

end

always@(posedge clk)

begin

if(rst)

begin

soc\_rd\_data <= 0;

end

else

begin

if(soc\_sel & (~soc\_wen))

begin

case(soc\_addr)

ARRD\_REG1: soc\_rd\_data <= reg1;

ARRD\_REG2: soc\_rd\_data <= reg2;

ARRD\_REG3: soc\_rd\_data <= reg3;

ARRD\_REG4: soc\_rd\_data <= reg4;

ARRD\_REG5: soc\_rd\_data <= w\_rd\_data\_out;

default: begin soc\_rd\_data <= reg1; end

endcase

end

end

end

memory1 memory1\_inst(

.clk(clk),

.cs(reg1[0]),

.wen(reg2[0]),

.rst(rst),

.addr(reg3[7:0]),

.wr\_data(reg4),

.rd\_data(w\_rd\_data\_out)

);

endmodule