Counter

module counter(input clk,

input rst,

input updown,

output reg [3:0] count

);

always@(posedge clk)

begin

if(rst)

count<=0;

else if(updown)

count=count+1;

else

count=count-1;

end

endmodule

module soc\_counter(

input clk,

input reset\_n,

input [31:0] addr,

input pselect,

input [31:0] pwdata,

input pwrite,

output reg [31:0] prdata

);

reg [1:0] reg1;

reg [1:0] reg2;

reg [1:0] reg3;

reg [31:0] reg4;

wire [3:0] count;

parameter REG1\_ADDR=32'd0,REG2\_ADDR=32'd4,REG3\_ADDR=32'd8,REG4\_ADDR=32'd12;

always@(posedge clk)

begin

if(!reset\_n)

begin

reg1<=0;

reg2<=0;

reg3<=0;

reg4<=0;

end

else

begin

if(pwrite & pselect)

begin

case(addr)

REG1\_ADDR: reg1 <= pwdata;

REG2\_ADDR: reg2 <= pwdata;

REG3\_ADDR: reg3 <= pwdata;

REG4\_ADDR: reg4 <= pwdata;

default: begin

reg1<= pwdata;

end

endcase

end

end

end

always@(posedge clk)

begin

if(!reset\_n)

begin

prdata <= 0;

end

else

begin

if((~pwrite) & pselect)

begin

case(addr)

REG1\_ADDR: prdata <= reg1;

REG2\_ADDR: prdata <= reg2;

REG3\_ADDR: prdata <= reg3;

REG4\_ADDR: prdata <= count;

default: begin

prdata <= reg1;

end

endcase

end

end

end

counter dut1(.clk(reg1[1:0]),.rst(reg2[1:0]),.updown(reg3[1:0]),.count(count));

endmodule

Tb

module tb\_soc\_counter();

reg clk;

reg rst\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

soc\_counter soc\_counter\_inst1(

.clk,

.rst\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

always #5 clk=~clk;

initial

begin

clk=0;

rst\_n=1;

addr <= 32'd0;

pwdata <= 32'd0;

end

initial begin

#10;

rst\_n=0;

#10;

pselect=1;pwrite=1;

#10;

pwdata=32'd0;addr = 32'd0;

#10;

pwdata=32'd0;addr = 32'd1;

#10;

pwrite = 0; addr = 32'd4;

end

endmodule

Readwrite

module readwrite(

input clk,

input reset\_n,

input [31:0] addr,

input pselect,

input [31:0] pwdata,

input pwrite,

output reg [31:0] prdata

);

reg [31:0] reg1;

reg [31:0] reg2;

reg [31:0] reg3;

reg [31:0] reg4;

reg [31:0] reg5;

parameter REG1\_ADDR=32'd0,REG2\_ADDR=32'd4,REG3\_ADDR=32'd8,REG4\_ADDR=32'd12,REG5\_ADDR=32'd16;

always@(posedge clk)

begin

if(!reset\_n)

begin

reg1<=0;

reg2<=0;

reg3<=0;

reg4<=0;

reg5<=0;

end

else

begin

if(pwrite & pselect)

begin

case(addr)

REG1\_ADDR: reg1 <= pwdata;

REG2\_ADDR: reg2 <= pwdata;

REG3\_ADDR: reg3 <= pwdata;

REG4\_ADDR: reg4 <= pwdata;

REG5\_ADDR: reg5 <= pwdata;

default: begin

reg1<= pwdata;

end

endcase

end

end

end

always@(posedge clk)

begin

if(!reset\_n)

begin

prdata <= 0;

end

else

begin

if((~pwrite) & pselect)

begin

case(addr)

REG1\_ADDR: prdata <= reg1;

REG2\_ADDR: prdata <= reg2;

REG3\_ADDR: prdata <= reg3;

REG4\_ADDR: prdata <= reg4;

REG5\_ADDR: prdata <= reg5;

default: begin

prdata <= reg1;

end

endcase

end

end

end

endmodule

Tb

module tb\_readwrite1();

reg clk;

reg reset\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

always #5 clk=~clk;

initial clk=0;

readwrite readwrite\_inst1(

.clk,

.reset\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

initial begin

reset\_n=0;

#20;

reset\_n=1;

#20;

pselect=1;pwrite=1;

#10;

pwdata=32'd2;addr = 32'd0;

#10;

pwdata=32'd4; addr = 32'd1;

#10;

pselect = 1; pwrite = 0; addr = 32'd2;

#10;

$finish;

end

endmodule

MUX

module mux41(

input [3:0]a,

input [1:0]s,

output reg [0:0]y

);

parameter s0=2'b00,s1=2'b01,s2=2'b10,s3=2'b11;

always@(\*)

begin

case(s)

s0: y <= a[0];

s1: y <= a[1];

s2: y <= a[2];

s3: y <= a[3];

endcase

end

endmodule

module soc\_mux(

input clk,

input reset\_n,

input [31:0] addr,

input pselect,

input [31:0] pwdata,

input pwrite,

output reg [31:0] prdata

);

reg [3:0] reg1;

reg [1:0] reg2;

reg [0:0] reg3;

wire [0:0] out;

parameter REG1\_ADDR=32'd0,REG2\_ADDR=32'd1,REG3\_ADDR=32'd2;

always@(posedge clk)

begin

if(!reset\_n)

begin

reg1<=0;

reg2<=0;

reg3<=0;

end

else

begin

if(pwrite & pselect)

begin

case(addr)

REG1\_ADDR: reg1 <= pwdata;

REG2\_ADDR: reg2 <= pwdata;

REG3\_ADDR: reg3 <= pwdata;

default: begin

reg1<= pwdata;

end

endcase

end

end

end

always@(posedge clk)

begin

if(!reset\_n)

begin

prdata <= 0;

end

else

begin

if((~pwrite) & pselect)

begin

case(addr)

REG1\_ADDR: prdata <= reg1;

REG2\_ADDR: prdata <= reg2;

REG3\_ADDR: prdata <= out;

default: begin

prdata <= reg1;

end

endcase

end

end

end

mux41 dut1(.a(reg1[3:0]),.s(reg2[1:0]),.y(out));

endmodule

TB

module tb\_soc\_mux();

reg clk;

reg reset\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

always #5 clk=~clk;

initial clk=0;

soc\_mux soc\_mux\_inst1(

.clk,

.reset\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

initial begin

reset\_n=0;

#20;

reset\_n=1;

#20;

pselect=1;pwrite=1;

#10;

pwdata=32'd6;addr = 32'd0;

#10;

pwdata=32'd0; addr = 32'd1;

#10;

pselect = 1; pwrite = 0; addr = 32'd2;

#10;pwrite=1;

#10

pwdata=32'd1; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd2; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd3; addr=32'd1;

#10;

$finish;

end

endmodule

Decoder

module tb\_soc\_mux();

reg clk;

reg reset\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

always #5 clk=~clk;

initial clk=0;

soc\_mux soc\_mux\_inst1(

.clk,

.reset\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

initial begin

reset\_n=0;

#20;

reset\_n=1;

#20;

pselect=1;pwrite=1;

#10;

pwdata=32'd6;addr = 32'd0;

#10;

pwdata=32'd0; addr = 32'd1;

#10;

pselect = 1; pwrite = 0; addr = 32'd2;

#10;pwrite=1;

#10

pwdata=32'd1; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd2; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd3; addr=32'd1;

#10;

$finish;

end

endmodule

Tb

module tb\_soc\_decoder();

reg clk;

reg reset\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

always #5 clk=~clk;

initial clk=0;

soc\_decoder soc\_decoder\_inst1(

.clk,

.reset\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

initial begin

reset\_n=0;

#20;

reset\_n=1;

#20;

pselect=1;pwrite=1;

#10;

pwdata=32'd6;addr = 32'd0;

#10;

pwdata=32'd0; addr = 32'd1;

#10;

pselect = 1; pwrite = 0; addr = 32'd2;

#10;

pwrite=1;

#10

pwdata=32'd1; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd2; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd3; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

#10

pwrite=1;

#10

pwdata=32'd2; addr=32'd1;

#10;

pwrite=0;

#10

pselect = 1; pwrite = 0; addr = 32'd2;

$finish;

end

endmodule

Adder

module adder1(

input [7:0]a,

input [7:0]b,

output [8:0]y

);

assign y=a+b;

endmodule

module soc\_adder1(

input clk,

input reset\_n,

input [31:0] addr,

input pselect,

input [31:0] pwdata,

input pwrite,

output reg [31:0] prdata

);

reg [31:0] reg1;

reg [31:0] reg2;

reg [31:0] reg3;

wire [8:0] sum;

parameter REG1\_ADDR=32'd0,REG2\_ADDR=32'd1,REG3\_ADDR=32'd2;

always@(posedge clk)

begin

if(!reset\_n)

begin

reg1<=0;

reg2<=0;

reg3<=0;

end

else

begin

if(pwrite & pselect)

begin

case(addr)

REG1\_ADDR: reg1 <= pwdata;

REG2\_ADDR: reg2 <= pwdata;

REG3\_ADDR: reg3 <= pwdata;

default: begin

reg1<= pwdata;

end

endcase

end

end

end

always@(posedge clk)

begin

if(!reset\_n)

begin

prdata <= 0;

end

else

begin

if((~pwrite) & pselect)

begin

case(addr)

REG1\_ADDR: prdata <= reg1;

REG2\_ADDR: prdata <= reg2;

REG3\_ADDR: prdata <= sum;

default: begin

prdata <= reg1;

end

endcase

end

end

end

adder1 dut1(.a(reg1[7:0]),.b(reg2[7:0]),.y(sum));

endmodule

Tb

module tb\_soc\_adder();

reg clk;

reg reset\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire[31:0] prdata;

always #5 clk=~clk;

initial clk=0;

soc\_adder1 soc\_adder1\_inst1(

.clk,

.reset\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

initial begin

reset\_n=0;

#20;

reset\_n=1;

#20;

pselect=1;pwrite=1;

#10;

pwdata=32'd0;addr = 32'd0;

#10;

pwdata=32'd7; addr = 32'd1;

#10;

pselect = 1; pwrite = 0; addr = 32'd2;

#10;

$finish;

end

Endmodule

Dflipflop

module dflipflop(input d,

input rst,

input clk,

output reg q);

always@(posedge clk)

if (!rst)

q <= 0;

else

q <= d;

endmodule

module soc\_dflipflop(

input clk,

input rst\_n,

input [31:0] addr,

input pselect,

input [31:0] pwdata,

input pwrite,

output reg prdata

);

reg [1:0] reg1;

reg reg2;

wire q;

parameter REG1\_ADDR=32'd0,REG2\_ADDR=32'd4;

always@(posedge clk)

begin

if(!rst\_n)

begin

reg1<=0;

reg2<=0;

end

else

begin

if(pwrite & pselect)

begin

case(addr)

REG1\_ADDR: reg1 <= pwdata;

REG2\_ADDR: reg2 <= pwdata;

default: begin

reg1<= pwdata;

end

endcase

end

end

end

always@(posedge clk)

begin

if(!rst\_n)

begin

prdata <= 32'b0;

end

else

begin

if((~pwrite) & pselect)

begin

case(addr)

REG1\_ADDR: prdata <= reg1;

REG2\_ADDR: prdata <= q ;

default: begin

prdata <= reg1;

end

endcase

end

end

end

dflipflop dut1(.rst(rst\_n),.clk(clk),.d(reg1[0]),.q(q));

endmodule

Tb

module tb\_soc\_dflipflop();

reg clk;

reg rst\_n;

reg [31:0] addr;

reg pselect;

reg [31:0] pwdata;

reg pwrite;

wire prdata;

soc\_dflipflop soc\_dflipflop\_inst1(

.clk,

.rst\_n,

.addr,

.pselect,

.pwdata,

.pwrite,

.prdata

);

always #5 clk=~clk;

initial

begin

clk=0;

rst\_n=0;

addr <= 32'd0;

pwdata <= 32'd0;

end

initial begin

#10;

rst\_n=1;

#10;

pselect=1;pwrite=1;

#10;

pwdata=32'd1;addr = 32'd0;

#10;

pwrite = 0;addr = 32'd4;

#500;

end

endmodule