**LAB 5 – Raja Aadhithan**

Design – Single port RAM:

Code:

module ram(input we\_in,enable\_in,

      input [3:0]addr\_in,

      inout [7:0]data);

   //Step1 : Declare a 8 bit wide memory having 16 locations.

reg [7:0] mem [15:0];

   //Understand the logic for writing data into a memory location

   always@(data,we\_in,enable\_in,addr\_in)

      if(we\_in && !enable\_in)

    mem[addr\_in]=data;

   //Understand the logic of reading data from a memory location

   assign data= (enable\_in && !we\_in) ? mem[addr\_in] : 8'hzz;

endmodule

Testbench:

module ram\_tb;

   wire [7:0] data;

   reg  [3:0] addr;

   reg  we,enable;

   reg  [7:0] tempd;

   integer l;

   //Step1 : Instantiate the RAM module and connect the ports

   ram dut(we,enable,addr,data);

   //Understand how the wire data acts like an input during write operation

   assign data=(we && !enable) ? tempd : 8'hzz;

    //Tasks for Initialising the inputs

   task initialize();

      begin

    we=1'b0; enable=1'b0; tempd=8'h00;

      end

   endtask

   /\*Step2 : Write a task named "stimulus" to assign data into

   "addr" and "tempd" inputs through i and j variables\*/

   task stimulus(input [3:0]i,

       input [7:0]j);

      begin

    addr = i;

    tempd = j;

      end

   endtask

   //Understand the various tasks used in this testbench

   task write();

      begin

    we=1'b1;

    enable=1'b0;

      end

   endtask

   task read();

      begin

    we=1'b0;

    enable=1'b1;

      end

   endtask

   task delay;

      begin

    #10;

      end

   endtask

   //Process to generate stimulus using for loop

   initial

      begin

    initialize;

    delay;

    write;

    for(l=0;l<16;l=l+1)

       begin

          stimulus(l,l);

          delay;

       end

    initialize;

    delay;

    read;

    for(l=0;l<16;l=l+1)

       begin

          stimulus(l,l);

          delay;

       end

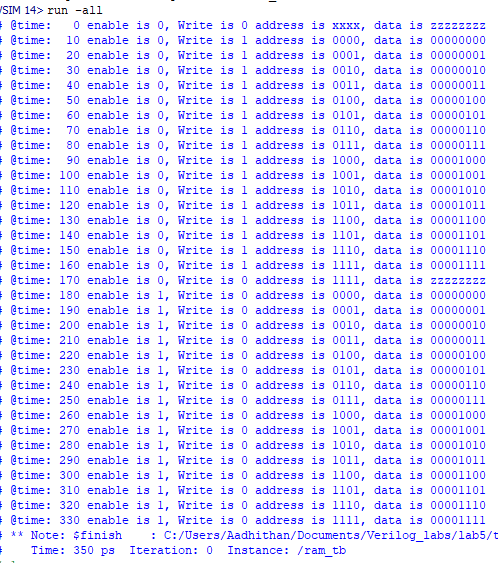
    delay;

    $finish;

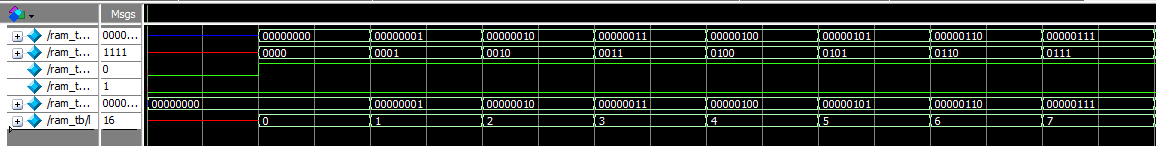
      end

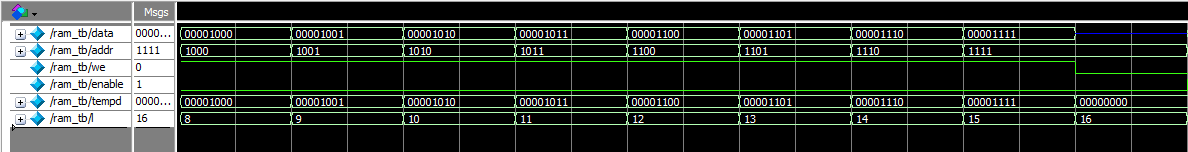
endmodule

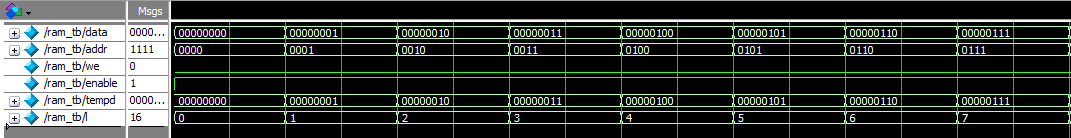
Output:

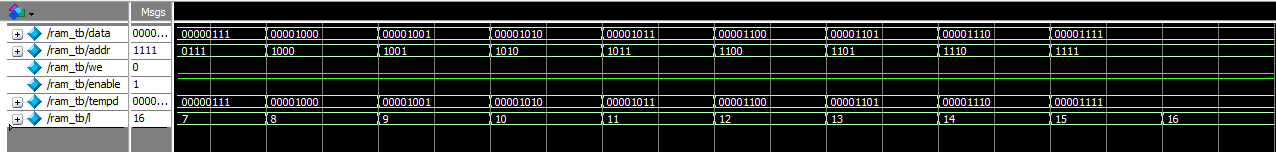


Wave:









RTL:

Too lengthy to crop.

**Exercise:**

Design : FIFO:

Code:

module fifo (clk,reset,read,write,data\_in,full,empty,data\_out);

parameter WIDTH = 8;

parameter DEPTH = 16;

output reg [WIDTH-1 : 0] data\_out;

output full;

output empty;

input [WIDTH-1 : 0] data\_in;

input clk;

input reset,read,write;

reg [WIDTH-1 : 0] mem [DEPTH-1 : 0];

reg [3 : 0] rd\_pointer;

reg [3 : 0] wr\_pointer;

assign empty = ((wr\_pointer - rd\_pointer) == 0) ? 1'b1 : 1'b0;

assign full  = ((wr\_pointer - rd\_pointer) == DEPTH) ? 1'b1 : 1'b0;

always @(posedge clk or negedge reset) begin

    if (!reset) begin

        // reset

        wr\_pointer <= 0;

        rd\_pointer <= 0;

    end

    else begin

        if (full == 1'b0 & write) begin

            mem[wr\_pointer] <= data\_in;

            wr\_pointer <= wr\_pointer + 1;

        end

        if (empty == 1'b0 & read) begin

            data\_out <= mem[rd\_pointer];

            rd\_pointer <= rd\_pointer + 1;

        end

    end

end

endmodule

Testbench:

module fifo\_tb();

reg clk,aresetn,read,write;

reg [7:0]data\_in;

wire full,empty;

wire [7:0]data\_out;

integer i;

fifo dut(clk,aresetn,read,write,data\_in,full,empty,data\_out);

initial begin

    clk = 1'b1;

    forever #5 clk = ~clk;

end

initial begin

    $monitor("$@time:%3d, read :%b, write :%b, datain = %b, dataout = %b",$time,read,write,data\_in,data\_out);

    aresetn = 0;

    #20;

    aresetn = 1;

    write = 1;

    read = 0;

    data\_in = 8'd5;

    #10;

    data\_in = 8'd15;

    #10;

    data\_in = 8'd21;

    #10;

    data\_in = 8'd31;

    #10;

    data\_in = 8'd41;

    #10;

    data\_in = 8'd12;

    #10;

    data\_in = 8'd13;

    #10;

    data\_in = 8'd33;

    #10;

    data\_in = 8'd42;

    #10;

    data\_in = 8'd16;

    #10;

    read = 1;

    data\_in = 8'd2;

    #10;

    data\_in = 8'd1;

    #10;

    data\_in = 8'd2;

    #10;

    data\_in = 8'd3;

    #10;

    data\_in = 8'd4;

    #10;

    data\_in = 8'd21;

    #10;

    data\_in = 8'd31;

    #10;

    data\_in = 8'd41;

    #10;

    data\_in = 8'd12;

    #10;

    data\_in = 8'd13;

    #10;

    data\_in = 8'd21;

    #10;

    data\_in = 8'd31;

    #10;

    data\_in = 8'd41;

    #10;

    data\_in = 8'd12;

    #10;

    data\_in = 8'd13;

    #10;

    write = 0;

    #30;

    read = 0;

    write =1;

    data\_in = 8'd10;

    #10;

    data\_in = 8'd11;

    #10;

    data\_in = 8'd21;

    #10;

    data\_in = 8'd31;

    #10;

    data\_in = 8'd41;

    #10;

    data\_in = 8'd12;

    #10;

    data\_in = 8'd13;

    #10;

    data\_in = 8'd33;

    #10;

    data\_in = 8'd42;

    #10;

    data\_in = 8'd16;

    #10;

    data\_in = 8'd22;

    #10;

    data\_in = 8'd30;

    #10;

    data\_in = 8'd06;

    #10;

    read = 1;

    write = 0;

    #35;

    write =1;

    data\_in = 8'd30;

    #10;

    data\_in = 8'd06;

    write =0;

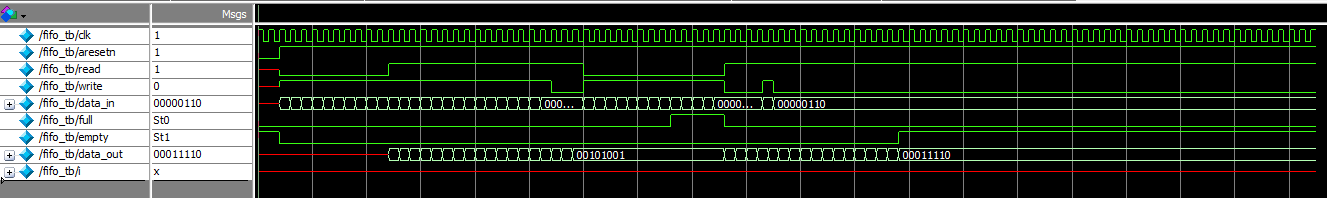
    #500;

    $finish;

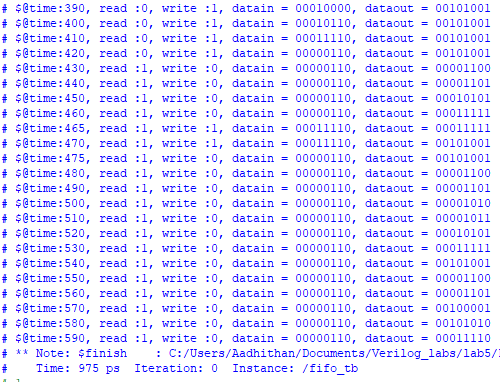
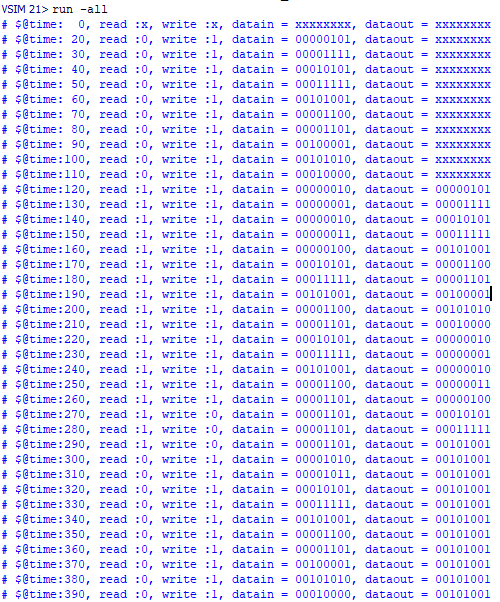
end

endmodule

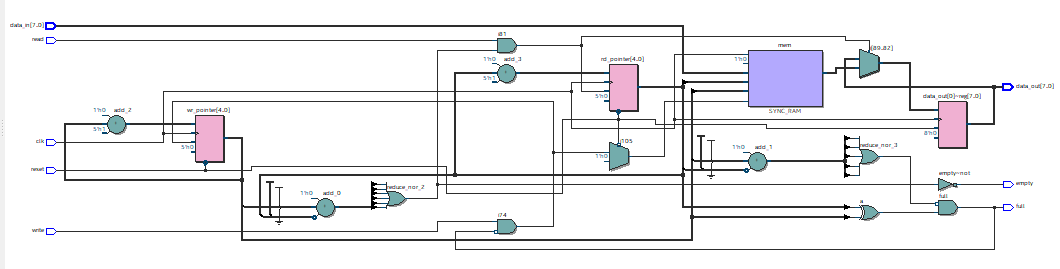
Wave: Zoomed out version



Output:



RTL:



Assignments:

Design : 16x8 dual port Ram

Code:

module ram168(input clk,wr\_in,rd\_in,

      input [3:0]addr\_in,addr\_out,

      input [7:0]data\_wr, output [7:0] data\_rd);

   reg [7:0]out;

reg [7:0] mem [15:0];

   always@(posedge clk)begin

      if(wr\_in) mem[addr\_in]=data\_wr;

      if(rd\_in) out = (rd\_in) ? mem[addr\_out] : 8'hzz;

   end

   assign data\_rd = out;

endmodule

Testbench:

module ram168\_tb();

reg clk,wr,rd;

reg [3:0] addr\_in,addr\_out;

reg [7:0]data\_wr;

wire [7:0] data\_rd;

integer i;

ram168 dut(clk,wr,rd,addr\_in,addr\_out,data\_wr,data\_rd);

initial begin

    clk =1;

    forever #5 clk = !clk;

end

initial begin

    wr = 1;

    rd = 0;

    $monitor("@time:%3d-wr:%b,%b,%b-rd:%b,%b,%b",$time,wr,addr\_in,data\_wr,rd,addr\_out,data\_rd);

    for (i=0;i<16;i=i+1)begin

        addr\_in = i[3:0];

        data\_wr = 2\*i+1;

        #15;

    end

    rd = 1;

    addr\_out = 14;

    #10;

    addr\_in = 4'd2;

    data\_wr = 8'd43;

    addr\_out = 4'd5;

    #10;

    addr\_in = 4'd6;

    data\_wr = 8'd31;

    addr\_out = 4'd7;

    #10;

    addr\_in = 4'd12;

    data\_wr = 8'd47;

    addr\_out = 4'd14;

    #10;

    addr\_in = 4'd1;

    data\_wr = 8'd4;

    addr\_out = 4'd1;

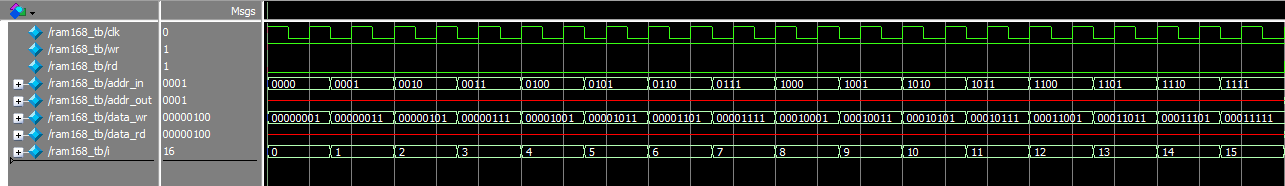
    #10;

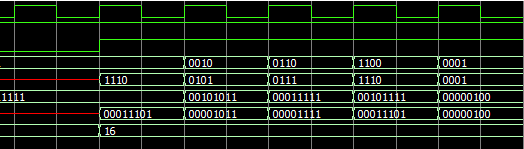
$finish;

end

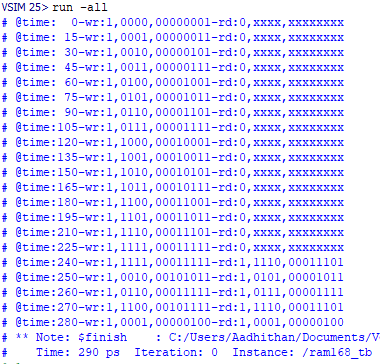
endmodule

Wave:

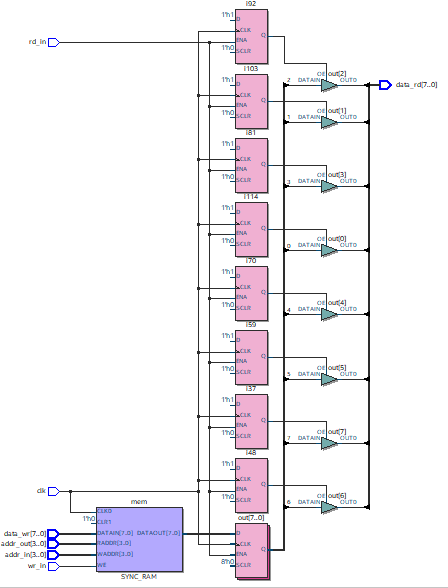




Output:



RTL:



Design : 8x16 Ram:

Code:

module aram8x16(input wr\_in,rd\_in,

      input [2:0]addr\_in,addr\_out,

      input [15:0]data\_wr, output [15:0] data\_rd);

   reg [7:0]out;

reg [15:0] mem [7:0];

   always@(\*)begin

      if(wr\_in) mem[addr\_in]=data\_wr;

      if(rd\_in) out = (rd\_in) ? mem[addr\_out] : 16'hzz;

   end

   assign data\_rd = out;

endmodule

Testbench:

module ram8x16\_tb();

reg wr,rd;

reg [2:0] addr\_in,addr\_out;

reg [15:0]data\_wr;

wire [15:0] data\_rd;

integer i;

aram8x16 dut(wr,rd,addr\_in,addr\_out,data\_wr,data\_rd);

initial begin

    wr = 1;

    rd = 0;

    $monitor("@time:%3d-wr:%b,%b,%b-rd:%b,%b,%b",$time,wr,addr\_in,data\_wr,rd,addr\_out,data\_rd);

    for (i=0;i<8;i=i+1)begin

        addr\_in = i[2:0];

        data\_wr = 2\*i+1;

        #15;

    end

    rd = 1;

    addr\_out = 7;

    #10;

    addr\_in = 3'd2;

    data\_wr = 16'd43;

    addr\_out = 3'd5;

    #10;

    addr\_in = 3'd6;

    data\_wr = 16'd31;

    addr\_out = 3'd7;

    #10;

    addr\_in = 3'd4;

    data\_wr = 16'd47;

    addr\_out = 3'd1;

    #10;

    addr\_in = 3'd1;

    data\_wr = 16'd114;

    addr\_out = 3'd1;

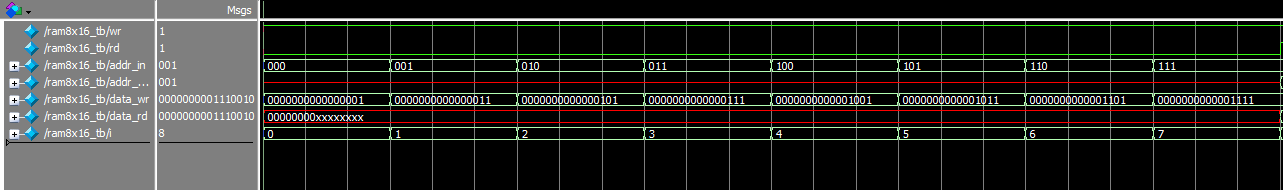
    #10;

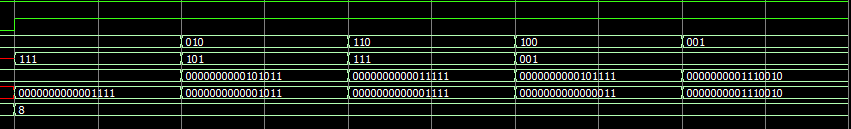
$finish;

end

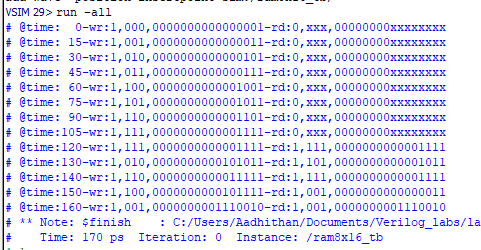
endmodule

Wave:





Output:



RTL:

Too length to attach.

Design : 4 bit SISO

Code:

module shift(input clk,data, output out);

reg [3:0] regis;

always@(posedge clk)begin

    regis[3] <= data;

    regis[2] <= regis[3];

    regis[1] <= regis[2];

    regis[0] <= regis[1];

end

assign out = regis[0];

endmodule

Testbench:

module shift\_tb();

reg clk,data;

wire out;

shift dut(clk,data,out);

initial begin

    clk = 1;

    forever #5 clk = ~clk;

end

initial begin

    $monitor("@time:%3d, in data is %b, out data is %b",$time,data,out);

    data = 1;

    #10;

    data = 1;

    #10;

    data = 0;

    #10;

    data = 1;

    #10;

    data = 0;

    #10;

    data = 0;

    #10;

    data = 1;

    #10;

    data = 0;

    #10;

    data = 1;

    #10;

    data = 1;

    #10;

    data = 0;

    #10;

    data = 1;

    #10;

    data = 0;

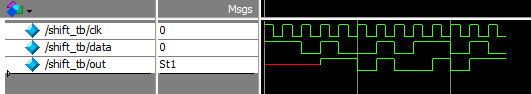
    #10;

$finish;

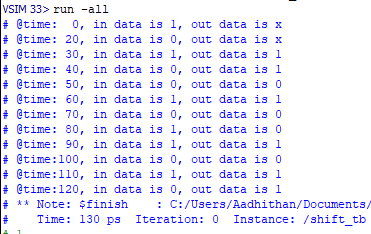
end

endmodule

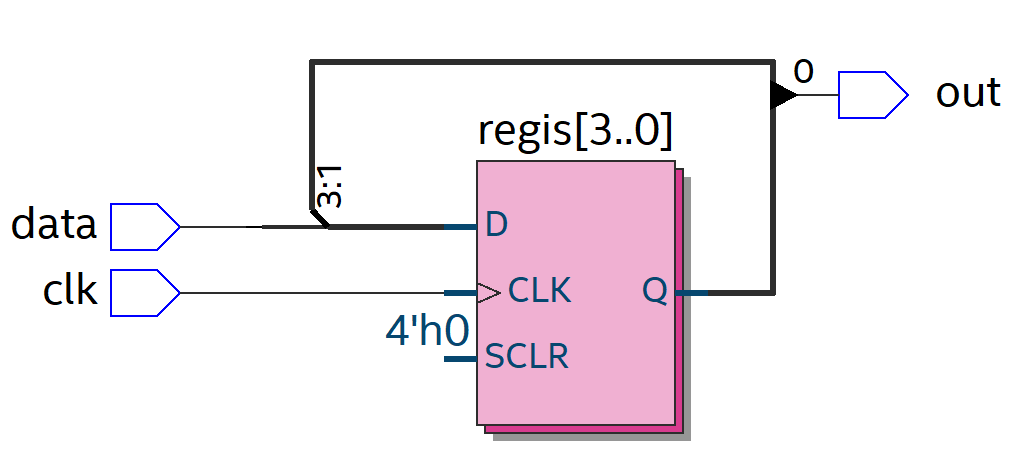
Wave:



Output:



RTL:



Design : Clock buffer:

Code:

module clockbuf(input clk,enb,output clk1,clk2,clk3,clk4);

wire x;

buf(x,clk);

bufif1(clk1,x,enb);

bufif1(clk2,x,enb);

bufif1(clk3,x,enb);

bufif1(clk4,x,enb);

endmodule

Testbench:

module clk\_tb();

reg clk,en;

wire clk1,clk2,clk3,clk4;

clockbuf dut(clk,en,clk1,clk2,clk3,clk4);

initial begin

    clk =1;

    forever #5 clk = ~clk;

end

initial begin

    $monitor("@time:%3d,in-clk:%b,out-clk:%b,%b,%b,%b",$time,clk,clk1,clk2,clk3,clk4);

    en = 0 ;

    #40;

    en = 1 ;

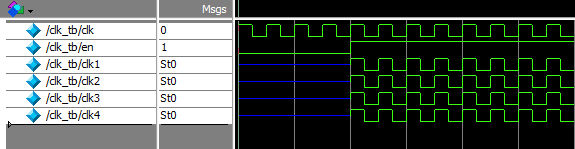
    #80;

    $finish;

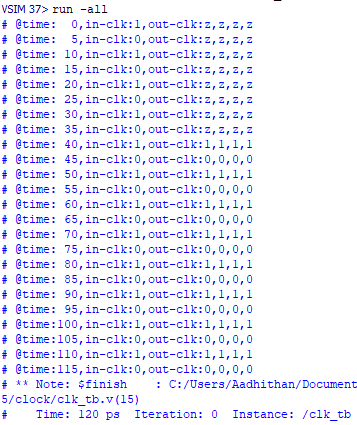
end

endmodule

Wave:



Output:



RTL:

