Part1. Multiplier Design

Implement a signed 4 bit sequential multiplier using Verilog. Use two four bit registers for the output of the multiplier (8 bit product).

```
module multiply(ready,plsb,prsb,product,multiplier,multiplicand,sign,clk);

  input        clk;
  input        sign;
  input [3:0]  multiplier, multiplicand;
  output [7:0] product;
   output [3:0] plsb,prsb;
  output       ready;

  reg [7:0]    product, product_temp;

  reg [3:0]    multiplier_copy;
  reg [7:0]    multiplicand_copy;
  reg          negative_output;

  reg [2:0]     bit;
  wire          ready = !bit;
   reg [3:0] plsb,prsb;

  initial bit = 0;
  initial negative_output = 0;

  always @( posedge clk )

    if( ready ) begin

      bit             = 3'b100;
      product         = 0;
      product_temp    = 0;
      multiplicand_copy = (!sign || !multiplicand[3]) ?
                        { 4'd0, multiplicand } :
                        { 4'd0, ~multiplicand + 1'b1};
      multiplier_copy   = (!sign || !multiplier[3]) ?
                        multiplier :
                        ~multiplier + 1'b1;

      negative_output = sign &&
                      ((multiplier[3] && !multiplicand[3])
                     ||(!multiplier[3] && multiplicand[3]));

    end
```

```
    else if ( bit > 0 ) begin

      if( multiplier_copy[0] == 1'b1 )
              product_temp = product_temp + multiplicand_copy;

      product = (!negative_output)?product_temp:(~product_temp + 1'b1);

      multiplier_copy = multiplier_copy >> 1;
      multiplicand_copy = multiplicand_copy << 1;
      bit = bit - 1'b1;

        plsb = product[3:0];
        prsb = product[7:4];

    end
endmodule
```

Part 2. Divider Design

Implement a sequential 4 bit divider using Verilog. Use two four bit registers as input and another two 4 bit registers to store quotient and reminder.

```
module sequential_divider(ready, quotient,reminder,dividend,divider,sign,clk);

  input        clk;
  input        sign;
  input [3:0]  dividend, divider;
  output [3:0]  quotient,reminder;
  output       ready;
  reg [3:0]    quotient, quotient_temp,dqb,drb;
  reg [7:0]    dividend_copy, divider_copy, diff;
  reg          negative_output;

  wire [3:0]   remainder = (!negative_output) ?
                            dividend_copy[3:0] :
                           ~dividend_copy[3:0] + 1'b1;
  reg [2:0]     bit;
  wire         ready = !bit;

  initial bit = 0;
  initial negative_output = 0;

  always @( posedge clk )
    if( ready ) begin
              bit = 3'd4;
              quotient = 0;
```

```
            quotient_temp = 0;
            dividend_copy = (!sign || !dividend[3]) ?
                            {4'd0,dividend} :
                            {4'd0,~dividend + 1'b1};
            divider_copy   =  (!sign || !divider[3]) ?
                            {1'b0,divider,3'd0} :
                            {1'b0,~divider + 1'b1,3'd0};
           negative_output = sign &&
                            ((divider[3] && !dividend[3])
                             ||(!divider[3] && dividend[3]));

    end
    else if ( bit > 0 ) begin

            diff = dividend_copy - divider_copy;
            quotient_temp = quotient_temp << 1;
            if( !diff[7] ) begin
                        dividend_copy = diff;
                       quotient_temp[0] = 1'd1;
            end
            quotient = (!negative_output) ?
                        quotient_temp :
                       ~quotient_temp + 1'b1;
            divider_copy = divider_copy >> 1;
            bit = bit - 1'b1;
    end
endmodule
```

Part 2:

Cpu_top.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:33:43 02/07/2010
// Design Name:
// Module Name:    cpu_top
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
```

```
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module cpu_top(clk, clear, inst_addr, instruction, r1, r2, r3, dp_result, comp_out)

input clk;
input clear;

output [3:0] inst_addr;
output [6:0] instruction;
output [3:0] r1, r2, r3;
output [3:0] dp_result;
output comp_out;

	wire cpu_clk, clk_1, clk_10, clk_100, clk_1k;

	wire simulation = 1'b1;
	assign cpu_clk = simulation ? clk : clk_1;
	gen_multi_clk c0(
		.CLOCK(clk),
		.CK_1Hz(clk_1),
		.CK_10Hz(clk_10),
		.CK_100Hz(clk_100),
		.CK_1KHz(clk_1k));

	wire [3:0] inst_addr;
	wire pc_enable, pc_load;
	wire comp_out;
	wire pc_en, r1_en, r1_sel, r2_en, clear_regfile, r3_en, comp_en, result_sel;
	wire [6:0] instruction;
	wire dp_result
	wire [3:0] r1, r2, r3

	//disable pc after address 1111
	assign pc_enable = ~(inst_addr[0] & inst_addr[1] & inst_addr[2] & inst_addr[3]);
	assign pc_load = pc_en & comp_out;
	program_counter pc0(
		.clk(cpu_clk),
		.enable(pc_enable),
		.reset(clear),
		.load(pc_load),
		.data(instruction[3:0]),
```

```
        .q(inst_addr));


    instruction_ROM ir0(
        .addr(inst_addr),
        .inst(instruction));



    instruction_decoder id0(
        .inst(instruction[6:4]),
        .pc_en(pc_en),
        .r1_en(r1_en),
        .r1_sel(r1_sel),
        .r2_en(r2_en),
        .clear(clear_regfile),
        .r3_en(r3_en),
        .comp_en(comp_en),
        .result_sel(result_sel))


    regfile r0(
        .clk(cpu_clk),
        .instant(instruction[3:0]),
        .dp_result(dp_result),
        .r1_en(r1_en),
        .r1_sel(r1_sel),
        .r2_en(r2_en),
        .clear(clear_regfile),
        .r3_en(r3_en),
        .comp_in(dp_comp_out),
        .comp_en(comp_en),
        .r1(r1),
        .r2(r2),
        .r3(r3),
        .comp_out(comp_out));


    datapath dp0(
        .r1(r1),
        .r2(r2),
        .op(result_sel),
        .comp(dp_comp_out),
        .dp_out(dp_result));

Endmodule
```

Datapath.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:26:37 02/07/2010
// Design Name:
// Module Name:    datapath
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module datapath(r1, r2, op, comp, dp_out)
input [3:0] r1;
input [3:0] r2;
input op;

output comp;
output [3:0] dp_out;

wire [7:0] op_result;

                assign op_result = op? (r1*r2):(r1+r2);
                assign dp_out = op_result[3:0];
                assign comp = (r1 <= r2);

endmodule
```

Instruction_decoder.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date:    21:54:43 02/07/2010
// Design Name:
// Module Name:    instruction_decoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module instruction_decoder(inst, pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en, result_sel)

input [2:0] inst;

output pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en, result_sel;

reg pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en, result_sel;

        always@(inst[2] or inst[1] or inst[0]) begin

                case(inst[2:0])
                //Initialization
                3'b000:begin
                                                pc_en <=0;
                                                r1_en <=1;
                                                r1_sel <=0;
                                                r2_en <=1;
                                                clear <=1;
                                                r3_en <=1;
                                                comp_en <=0;
                                                result_sel <=0;
                                        end
                //Move instant number to r1
                3'b001:  begin
                                                pc_en <=0
                                                r1_en <=1
                                                r1_sel <=0
                                                r2_en <=0
                                                clear <=0
                                                r3_en <=0
                                                comp_en <=0
```

```
                                result_sel <=0;
                        end
//Move instant number to r2
3'b010:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=1
                                clear <=0
                                r3_en <=0
                                comp_en <=0
                                result_sel <=0
                        end
//Move previous r3 value to r1
3'b011:  begin
                                pc_en <=0
                                r1_en <=1
                                r1_sel <=1
                                r2_en <=0
                                clear <=0
                                r3_en <=0
                                comp_en <=0
                                result_sel <=0
                        end
//Add: r3=r1+r2
3'b100:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=0
                                clear <=0
                                r3_en <=1
                                comp_en <=0
                                result_sel <=0
                        end
//Multiplication: r3=r1r2
3'b101:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=0
                                clear <=0
                                r3_en <=1
                                comp_en <=0
                                result_sel <=1
                        end
```

```
                //Comparison the result of r1 <= r2
                3'b110:  begin
                                        pc_en <=0
                                        r1_en <=0
                                        r1_sel <=0
                                        r2_en <=0
                                        clear <=0
                                        r3_en <=0
                                        comp_en <=1
                                        result_sel <=0
                                end
                //Branch if comp_result == 1
                3'b111:  begin
                                        pc_en <=1
                                        r1_en <=0
                                        r1_sel <=0
                                        r2_en <=0
                                        clear <=0
                                        r3_en <=0
                                        comp_en <=1
                                        result_sel <=0
                                end

                endcase
        end

endmodule
```

Program_counter.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:41:38 02/07/2010
// Design Name:
// Module Name:    program_counter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
```

```
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module program_counter(clk, enable, reset, load, data, q)

input clk, enable, reset, load;
input [3:0] data;

output [3:0] q;
reg [3:0] q;
	always@(posedge clk) begin

	if(reset)
		q <= 0;
	else if(load)
		q <= data;
	else if(enable)
		q <= q+1;
	end

endmodule
```

regfile.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:13:22 02/07/2010
// Design Name:
// Module Name:    regfile
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
```

```
module regfile(clk, instant, dp_result, r1_en, r1_sel, r2_en, clear, r3_en, comp_in, comp_en, r1,
r2, r3, comp_out)
input clk, r1_en, r1_sel, r2_en, clear, r3_en, comp_in, comp_en;
input [3:0] instant;
input [3:0] dp_result;

output [3:0] r1;
output [3:0] r2;
output [3:0] r3;
output comp_out

reg [3:0] r1
reg [3:0] r2
reg [3:0] r3
reg comp_out;

        always@(posedge clk) begin
                if(clear) begin
                        r1 <=4'b0000;
                        r2 <=4'b0000;
                        r3 <=4'b0000;
                        comp_out <=1'b0;
                end

                else if(r1_en) begin
                        r1 <= r1_sel ? r3:instant;
                end

                else if(r2_en) begin
                        r2 <= instant;
                end

                else if(r3_en) begin
                        r3 <=dp_result;
                end

                else if(comp_en) begin
                        comp_out <= comp_in;
                end

        end

endmodule


// dffre: D flip-flop with active high enable and reset
```

```
// Parametrized width; default of 1
module dffre
(
d,
en,
r,
clk,
q
);
parameter WIDTH = 1;
input en;
input r;
input clk;
input [WIDTH-1:0] d;
output [WIDTH-1:0] q;
reg [WIDTH-1:0] q;
always @ (posedge clk)
if ( r )
q <= {WIDTH{1'b0}};
else if (en)
q <= d;
else q <= q;
endmodule
```

Part 3

Add divide instruction to your CPU from part 2, use R1 and R2 as input registers , use R3 as quotient R4 for the remainder of division.

Cpu_top.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:33:43 02/07/2010
// Design Name:
// Module Name:    cpu_top
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
```

```
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module cpu_top(clk, clear, inst_addr, instruction, r1, r2, r3,r4, dp_result, comp_out,dp_result1)

input clk;
input clear;

output [3:0] inst_addr;
output [7:0] instruction;
output [3:0] r1, r2, r3,r4;
output [3:0] dp_result,dp_result1;
output comp_out;

	wire cpu_clk, clk_1, clk_10, clk_100, clk_1k;

	wire simulation = 1'b1;
	assign cpu_clk = simulation ? clk : clk_1;
	gen_multi_clk c0(
		.CLOCK(clk),
		.CK_1Hz(clk_1),
		.CK_10Hz(clk_10),
		.CK_100Hz(clk_100),
		.CK_1KHz(clk_1k));

	wire [3:0] inst_addr;
	wire pc_enable, pc_load;
	wire comp_out;
	wire pc_en, r1_en, r1_sel, r2_en, clear_regfile, r3_en, comp_en,r4_en;
	wire [1:0] result_sel;
	wire [7:0] instruction;
	wire dp_result, dp_result1;
	wire [3:0] r1, r2, r3,r4

	//disable pc after address 1111
	assign pc_enable = ~(inst_addr[0] & inst_addr[1] & inst_addr[2] & inst_addr[3]);
	assign pc_load = pc_en & comp_out;
	program_counter pc0(
		.clk(cpu_clk),
		.enable(pc_enable),
		.reset(clear),
		.load(pc_load),
		.data(instruction[3:0]),
		.q(inst_addr));
```

```
instruction_ROM ir0(
        .addr(inst_addr),
        .inst(instruction));



instruction_decoder id0(
        .inst(instruction[7:4]),
        .pc_en(pc_en),
        .r1_en(r1_en),
        .r1_sel(r1_sel),
        .r2_en(r2_en),
        .clear(clear_regfile),
        .r3_en(r3_en),
        .comp_en(comp_en),
        .result_sel(result_sel),
        .r4_en(r4_en));


regfile r0(
        .clk(cpu_clk),
        .instant(instruction[3:0]),
        .dp_result(dp_result),
        .r1_en(r1_en),
        .r1_sel(r1_sel),
        .r2_en(r2_en),
        .clear(clear_regfile),
        .r3_en(r3_en),
        .comp_in(dp_comp_out),
        .comp_en(comp_en),
        .r1(r1),
        .r2(r2),
        .r3(r3),
        .comp_out(comp_out)
        .r4_en(r4_en),
        .r4(r4),
        .dp_result1(dp_result1));


datapath dp0(
        .r1(r1),
        .r2(r2),
        .op(result_sel),
        .comp(dp_comp_out),
```

```
                .dp_out(dp_result),
                .dp_out1(dp_result1));

Endmodule
```

Datapath.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:26:37 02/07/2010
// Design Name:
// Module Name:    datapath
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module datapath(r1, r2, op, comp, dp_out,dp_out1)
input [3:0] r1;
input [3:0] r2;
input[1:0] op;

output comp;
output [3:0] dp_out,dp_out1;

reg [7:0] op_result,op_result1;

                always @ (op)
                        begin
                                if(op == 2'b00)
                                        begin
                                                op_result = r1*r2;
                                                op_result1 = 0;
                                        end
                                if(op == 2'b01)
                                        begin
```

```
                                        op_result = r1 + r2;
                                        op_result1 = 0;
                                end
                        if(op == 2'b10)
                                begin
                                        op_result = r1/r2;
                                        op_result1 = r1%r2;
                                end
                end
        assign dp_out = op_result[3:0];
        assign dp_out1 = op_result1[3:0];
        assign comp = (r1 <= r2);

endmodule
```

Instruction_decoder.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:54:43 02/07/2010
// Design Name:
// Module Name:    instruction_decoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module instruction_decoder(inst, pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en,
result_sel,r4_en)

input [3:0] inst;

output pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en, result_sel,r4_en;
output reg [1:0] result_sel;
reg pc_en, r1_en, r1_sel, r2_en, clear, r3_en, comp_en,r4_en;
```

```
always@(inst[3] or inst[2] or inst[1] or inst[0]) begin

        case(inst[3:0])
        //Initialization
        4'b0000:begin
                                        pc_en <=0;
                                        r1_en <=1;
                                        r1_sel <=0;
                                        r2_en <=1;
                                        clear <=1;
                                        r3_en <=1;
                                        r4_en <=0;
                                        comp_en <=0;
                                        result_sel <=0;
                                end
        //Move instant number to r1
        3'b0001:  begin
                                        pc_en <=0
                                        r1_en <=1
                                        r1_sel <=0
                                        r2_en <=0
                                        clear <=0
                                        r3_en <=0
                                        r4_en <=0;
                                        comp_en <=0
                                        result_sel <=0;
                                end
        //Move instant number to r2
        4'b0010:  begin
                                        pc_en <=0
                                        r1_en <=0
                                        r1_sel <=0
                                        r2_en <=1
                                        clear <=0
                                        r3_en <=0
                                        r4_en <=0;
                                        comp_en <=0
                                        result_sel <=0
                                end
        //Move previous r3 value to r1
        4'b0011:  begin
                                        pc_en <=0
                                        r1_en <=1
                                        r1_sel <=1
                                        r2_en <=0
```

```
                                clear <=0
                                r3_en <=0
                                r4_en <=0
                                comp_en <=0
                                result_sel <=0
                        end
//Add: r3=r1+r2
4'b0100:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=0
                                clear <=0
                                r3_en <=1
                                comp_en <=0
                                result_sel <=0
                                r4_en <=0
                        end
//Multiplication: r3=r1r2
4'b0101:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=0
                                clear <=0
                                r3_en <=1
                                comp_en <=0
                                result_sel <=1
                                r4_en <=0
                        end
//Comparison the result of r1 <= r2
4'b0110:  begin
                                pc_en <=0
                                r1_en <=0
                                r1_sel <=0
                                r2_en <=0
                                clear <=0
                                r3_en <=0
                                comp_en <=1
                                result_sel <=0
                                r4_en <=0
                        end
//Branch if comp_result == 1
4'b0111:  begin
                                pc_en <=1
                                r1_en <=0
```

```
                                        r1_sel <=0
                                        r2_en <=0
                                        clear <=0
                                        r3_en <=0
                                        comp_en <=1
                                        result_sel <=0
                                        r4_en <=0
                                end
                //Division
                4'b1000:  begin
                                        pc_en <=0
                                        r1_en <=0
                                        r1_sel <=0
                                        r2_en <=0
                                        clear <=0
                                        r3_en <=1
                                        comp_en <=0
                                        result_sel <=2
                                        r4_en <=1
                                end


                endcase
        end

endmodule
```

Program_counter.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:41:38 02/07/2010
// Design Name:
// Module Name:    program_counter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
```

```
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module program_counter(clk, enable, reset, load, data, q)

input clk, enable, reset, load;
input [3:0] data;

output [3:0] q;
reg [3:0] q;
	always@(posedge clk) begin

	if(reset)
		q <= 0;
	else if(load)
		q <= data;
	else if(enable)
		q <= q+1;
	end

endmodule
```

regfile.v

```
`timescale 1ns / 1p
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:13:22 02/07/2010
// Design Name:
// Module Name:    regfile
// Project Name:
// Target Devices:
// Tool versions:
// Description:
/
// Dependencies:
/
// Revision:
// Revision 0.01 - File Create
// Additional Comments:
/
//////////////////////////////////////////////////////////////////////////////////
module regfile(clk, instant, dp_result, r1_en, r1_sel, r2_en, clear, r3_en, comp_in, comp_en, r1,
r2, r3, comp_out,r4_en,r4,dp_result1)
```

```
input clk, r1_en, r1_sel, r2_en, clear, r3_en, comp_in, comp_en,r4_en;
input [3:0] instant;
input [3:0] dp_result;
input [3:0] dp_result1;

output [3:0] r1;
output [3:0] r2;
output [3:0] r3;
output [3:0] r4;
output comp_out;

reg [3:0] r1;
reg [3:0] r2;
reg [3:0] r3;
reg[3:0]  r4;
reg comp_out;

        always@(posedge clk) begin
                if(clear) begin
                        r1 <=4'b0000;
                        r2 <=4'b0000;
                        r3 <=4'b0000;
                        r4 <=4’b0000;
                        comp_out <=1'b0;
                end

                else if(r1_en) begin
                        r1 <= r1_sel ? r3:instant;
                end

                else if(r2_en) begin
                        r2 <= instant;
                end

                else if(r3_en) begin
                        r3 <=dp_result;
                end

                else if(comp_en) begin
                        comp_out <= comp_in;
                end

                if(r4_en && !clear) begin
                        r4 <= dp_result1;
                end
```

```
        end

endmodule
```