## VLSI Lab 2 – EDM39 & COE03

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 1 - ALU
// OPCODE: FUNCTION
////////////////

//000: Add = a + b + c_in
//
//001: Sub = a - b + c_in
//
//010: Subc = b - a - c_in
//
//011: Or = 1'b0, a|b
//
//100: And = 1'b0, a & b
//
//101: Not = 1'b0, ~a & b
//
//110: Exor = 1'b0, a^b
//
//111: Exnor = 1'b0, a~^b

//////////////////////////////////////////////////////////////////////////////////
module ALU(
    input [7:0] a,
    input [7:0] b,
    input c_in,
    input [2:0] oper,
    output reg [7:0] res,
    output reg c_out
    );
        reg [7:0] bComp;
        reg [7:0] c_inBus = 8'd0;
        reg [7:0] c_inComp;
        always @(*) begin
                c_out = 1'b0;
                bComp = (~b) + 8'd1;
                case(oper)
                        4'h0: begin //ADD
                                {c_out, res} = a + b;
                                {c_out, res} = {c_out, res} + c_in;
                        end
                        4'h1: begin //SUB
                                {c_out, res} = a + bComp; //Adding 2s Complement
                                {c_out, res} = {c_out, res} + c_in;
                        end
                        4'h2: begin //Subc
                                c_inBus[0] = c_in;
                                c_inComp = (~c_inBus) + 8'd1;
                                {c_out, res} = a + bComp; //Adding 2s Complement
                                {c_out, res} = {c_out, res} + c_inComp;
                        end
                        4'h3: begin //BITOR
                                res = a | b;
                        end
                        4'h4: begin //BITAND (&)
                                res = a & b;
                        end
                        4'h5: begin //NOT
                                res = ~a;
                        end
                        4'h6: begin //BITXOR (^)
                                res = a ^ b;
                        end
                        4'h7: begin //BITXNOR (~)
                                res = a ~^ b;
                        end
                endcase
        end
endmodule
```

```
`timescale 1ns / 1ns
////////////////////////////////////////////////////////////////////////////////
// Qustion 1 - ALU - Test Bench
//
//OPCODE: FUNCTION
////////////////
//000: Add = a + b + c_in
//001: Sub = a - b + c_in
//010: Subc = b - a - c_in
//011: Or = 1'b0, a|b
//100: And = 1'b0, a & b
//101: Not = 1'b0, ~a & b
//110: Exor = 1'b0, a^b
//111: Exnor = 1'b0, a~^b
////////////////////////////////////////////////////////////////////////////////
module ALU_tb;

	// Inputs
	reg [7:0] a;
	reg [7:0] b;
	reg c_in;
	reg [2:0] oper;

	// Outputs
	wire [7:0] res;
	wire c_out;


	integer i;

	// Instantiate the Unit Under Test (UUT)
	ALU uut (
		.a(a),
		.b(b),
		.c_in(c_in),
		.oper(oper),
		.res(res),
		.c_out(c_out)
	);


	initial begin
		// Initialize Inputs
		a = 0;
		b = 0;
		c_in = 0;
		oper = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		a = 8'd200;
		b = 8'd50;
		c_in = 1'b0;
		oper = 3'd0;


		for(i=0; i < 8; i=i+1) begin
			#5 oper = oper + 3'd1;
		end

		#5 $finish;

	End
```

```
Endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
/*
	Question 2
	Find max of 4 32-bit unsigned numbers by subtracting 33-bit signed numbers and finding
	sign bit.
*/
//////////////////////////////////////////////////////////////////////////////
module C_4_32(
    input [31:0] a,
    input [31:0] b,
    input [31:0] c,
    input [31:0] d,
    output reg [31:0] max
    );
        reg [32:0] as, bs, cs, ds;
        reg [32:0] bsc, dsc;
        reg [32:0] cmpab;
        reg [31:0] maxab;
        reg [32:0] cmpcd;
        reg [31:0] maxcd;
        reg [32:0] maxabs, maxcds, maxcdc;
        reg [32:0] cmp_ab_cd;
        always @(*) begin
                as = {1'b0, a};
                bs = {1'b0, b};
                cs = {1'b0, c};
                ds = {1'b0, d};
                bsc = (~bs) + 33'd1;
                dsc = (~ds) + 33'd1;
                cmpab = as + bsc;

                if(cmpab[32]) begin
                        maxab = b;
                end
                else begin
                        maxab = a;
                end
                cmpcd = cs + dsc;
                if(cmpcd[32]) begin
                        maxcd = d;
                end
                else begin
                        maxcd = c;
                end
                maxabs = {1'b0, maxab};
                maxcds = {1'b0, maxcd};
                maxcdc = (~maxcds) + 33'd1;
                cmp_ab_cd = maxabs + maxcdc;
                if(cmp_ab_cd[32]) begin
                        max = maxcd;
                end
                else begin
                        max = maxab;
                end
        end
endmodule
```

```
`timescale 1ns / 500ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 3 - divide_by_13
// Dependencies:
//////////////////////////////////////////////////////////////////////////////////
module divide_by_13(
    input in_clk,
    input reset,
    output out_clk
    );
        reg [3:0] count = 4'd0;
        reg flag = 0;
        assign out_clk = count[3];

        always @ (posedge in_clk) begin

                if (~reset) begin

                        if(flag) begin
                                if(count != 4'd12) begin
                                        count <= count + 1;
                                end
                                else begin
                                        count <= 4'd0;
                                end
                        end
                        else begin
                                flag <= 1'b1;
                        end
                end

                else begin
                        count <= 4'd0;
                        flag <= 1'b0;
                end
        end
endmodule

`timescale 1ns / 500ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 3 - divide_by_256
// Dependencies:
//////////////////////////////////////////////////////////////////////////////////
module divide_by_256(
    input in_clk,
    input reset,
    output [7:0] out_clks
    );
        reg [7:0] count = 8'd0;
        reg flag = 0;
        assign out_clks = count;

        always @ (posedge in_clk) begin
                if (~reset) begin
                        if(flag) begin
                                count <= count + 1;
                        end
                        else begin
                                flag <= 1'b1;
                        end
                end
                else begin
                        count <= 8'd0;
                        flag <= 1'b0;
                end
        end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 3 - mux_8_1
// Dependencies:
//////////////////////////////////////////////////////////////////////////////////
module mux_8_1(
    input [7:0] inp,
    input [2:0] sel,
    output reg out
    );

        always @ (*) begin
                case(sel)
                        3'd0:
                                out = inp[0];
                        3'd1:
                                out = inp[1];
                        3'd2:
                                out = inp[2];
                        3'd3:
                                out = inp[3];
                        3'd4:
                                out = inp[4];
                        3'd5:
                                out = inp[5];
                        3'd6:
                                out = inp[6];
                        3'd7:
                                out = inp[7];
                endcase
        end

endmodule



`timescale 1ns / 500ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 3 - divide_by_8
// Dependencies:
//////////////////////////////////////////////////////////////////////////////////
module divide_by_8(
    input in_clk,
    input reset,
    output out_clk
    );

        reg [2:0] count = 3'd0;
        reg flag = 0;

        assign out_clk = count[2];

        always @ (posedge in_clk) begin

                if (~reset) begin
                        if(flag) begin
                                count <= count + 1;
                        end
                        else begin
                                flag <= 1'b1;
                        end
                end
                else begin
                        count <= 3'd0;
                        flag <= 1'b0;
                end
        end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Qustion 3 - mux_8_1
// Dependencies:
//        divide_by_13
//        divide_by_256
//        mux_8_1
//        divide_by_8
//////////////////////////////////////////////////////////////////////////////////
module UART_Clock_Gen(
    input sys_clk,
    input reset,
    input [2:0] sel_baud_rate,
    output clock,
    output sample_clk
    );

        wire clk_by_13;
        wire [7:0] clks_by_256;


        divide_by_13 d13(sys_clk, reset, clk_by_13);
        divide_by_256 d256(clk_by_13, reset, clks_by_256);
        mux_8_1 m(clks_by_256, sel_baud_rate, clock);
        divide_by_8 d8(clock, reset, sample_clk);

endmodule
```

## VLSI Lab 3 – EDM39 & COE03

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Question 1 - Synchronous 16 x 4 Memory (using clock) activated by chip select
//////////////////////////////////////////////////////////////////////////////////
module mem(
    input chipSel,
    input clk,
    input [3:0] wrDat,
    output reg [3:0] reDat,
    input [3:0] wrDat_addr,
    input [3:0] reDat_addr,
    input write
    );

    reg [3:0] DM [0:15];


    always @ (posedge clk) begin
      if(chipSel) begin
          if(write) begin

              if(wrDat_addr <= 15 && wrDat_addr >= 0)
                  DM[wrDat_addr] <= wrDat;
              else
                  $display("Error in wrDat_addr...");
          end

          if(reDat_addr <= 15 && reDat_addr >= 0)
                  reDat <= DM[reDat_addr];
          else
              $display("Error in reDat_addr...");
      end
    end
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Question 1 (TestBench) - Synchronous 16 x 4 Memory (using clock) activated by chip select
////////////////////////////////////////////////////////////////////////////////
module mem_tb;
	// Inputs
	reg chipSel;
	reg clk;
	reg [3:0] wrDat;
	reg [3:0] wrDat_addr;
	reg [3:0] reDat_addr;
	reg write;
	// Outputs
	wire [3:0] reDat;
	// Instantiate the Unit Under Test (UUT)
	mem uut (
		.chipSel(chipSel),
		.clk(clk),
		.wrDat(wrDat),
		.reDat(reDat),
		.wrDat_addr(wrDat_addr),
		.reDat_addr(reDat_addr),
		.write(write)
	);
	initial begin
		// Initialize Inputs
		chipSel = 1'b1;
		clk = 0;
		wrDat = 0;
		wrDat_addr = 0;
		reDat_addr = 0;
		write = 0;
		// Wait 100 ns for global reset to finish
		#100;
		// Add stimulus here
		#2 wrDat = 4'd15;
		#1 write = 1'b1;
		#1 wrDat_addr = 4'd1;
		#1;

		//clk posedge
		#5;

		//clk negedge
		#2 wrDat_addr = 4'd2;
		#1 wrDat = 4'd14;

		#2;
		//clk posedge

		#1 write = 1'b0;

		#8 reDat_addr = 4'd1;
		#1; //posedge clk

		#9 reDat_addr = 4'd1;
		#1; //posedge clk

		#9 reDat_addr = 4'd2;
		#1; //posedge clk

		#10 $finish;
	end

	always begin
		#5 clk = ~clk;
	end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Question 2 - Pseudo Random Number Generator using a Linear Feedback Shift Register
// 8 bit number generated between 1-255 and then MSB is forced to 0 to yield an output in the
range 0-127
//////////////////////////////////////////////////////////////////////////////////
module prng_8b_128(
    input clk,
    input reset,
    output [7:0] num
    );

        reg flag = 1'b0;
        reg [7:0] count = 8'd128;

        assign num[7] = 1'b0;
        assign num[6:0] = count[6:0];

        assign newbit = count[7]^count[5]^count[4]^count[3];

        always @ (posedge clk) begin

                if (~reset) begin
                        if(flag) begin
                                count <= count << 1;
                                count[0] <= newbit;
                        end
                        else begin
                                flag <= 1'b1;
                        end
                end

                else begin
                        count <= 8'd128;
                        flag <= 1'b0;
                end
        end

endmodule
```