EXPERIMENT- 7
Design and Verilog modeling of Euclid GCD Algorithm

Objective:

Design and Verilog modeling of datapath and control path of GCD algorithm using any modeling style.

Exercise Problems

1. **Design a datapath for dividing two numbers input by the user, using adder/subtraction, comparator units. Identify the high-level state diagram for the machine.**

Source Code

```
module Division (
    input clk, rst, start,
    input [7:0] dividend, divisor,
    output [7:0] quotient, remainder,
    output done
  );
  wire load_dividend, load_divisor, load_quotient, subtract;
  wire div_geq_div;
  DP dp (
       .clk(clk), .rst(rst),
       .dividend_in(dividend), .divisor_in(divisor),
       .load_dividend(load_dividend), .load_divisor(load_divisor),
       .load_quotient(load_quotient), .subtract(subtract),
       .quotient(quotient), .remainder(remainder),
       .div_geq_div(div_geq_div),.done(done)
     );
  CP cp (
       .clk(clk), .rst(rst), .start(start),
       .div_geq_div(div_geq_div),
       .load_dividend(load_dividend), .load_divisor(load_divisor),
       .load_quotient(load_quotient), .subtract(subtract),
       .done(done)
     );
endmodule
```

```
module CP (
    input clk, rst, start,
    input div_geq_div,
    output reg load_dividend, load_divisor, load_quotient,
    output reg subtract, done
  );
  parameter IDLE = 3'b000, LOAD = 3'b001, COMPARE = 3'b010,
            SUBTRACT = 3'b011, DONE = 3'b100;

  reg [2:0] state, next_state;
  always @(posedge clk or posedge rst)
  begin
    if (rst)
      state <= IDLE;
    else
      state <= next_state;
  end
  always @(posedge clk)
  begin
    load_dividend = 0;
    load_divisor = 0;
    load_quotient = 0;
    subtract = 0;
    done = 0;
    case (state)
      IDLE:
      begin
        if (start)
          next_state = LOAD;
        else
          next_state = IDLE;
      end
      LOAD:
      begin
        load_dividend = 1;
        load_divisor = 1;
        load_quotient = 1;
        next_state = COMPARE;
      end
      COMPARE:
      begin
        if (div_geq_div)
```

```
          next_state = SUBTRACT;
        else
          next_state = DONE;
      end
      SUBTRACT:
      begin
        subtract = 1;
        next_state = COMPARE;
      end
      DONE:
      begin
        done = 1;
        next_state = IDLE;
      end
      default:
        next_state = IDLE;
    endcase
  end

endmodule
module DP (
    input clk, rst,
    input [7:0] dividend_in, divisor_in,
    input load_dividend, load_divisor, load_quotient,
    input subtract,
    input done,
    output reg [7:0] quotient,
    output reg [7:0] remainder,
    output reg div_geq_div
  );
  reg [7:0] dividend, divisor;
  always @(posedge clk or posedge rst)
  begin
    if (rst)
    begin
      dividend <= 8'b0;
      divisor <= 8'b0;
      quotient <= 8'b0;
      remainder <= 8'b0;
    end
    else
    begin
```

```
      if (load_dividend)
        dividend <= dividend_in;
      if (load_divisor)
        divisor <= divisor_in;
      if (load_quotient)
        quotient <= 8'b0;
    end
  end
  always @(*)
  begin
    div_geq_div = (dividend >= divisor);
  end
  always @(posedge clk)
  begin
    if (subtract && div_geq_div)
    begin
      dividend <= dividend - divisor;
      quotient <= quotient + 1;
    end
    if (done==1)
      remainder <= dividend;
  end
endmodule
```

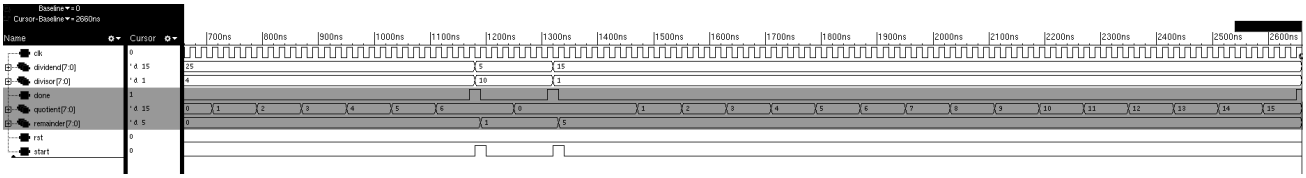
Test-bench

```
module tb();
  reg clk, rst, start;
  reg [7:0] dividend, divisor;
  wire [7:0] quotient, remainder;
  wire done;
  Division uut (
              .clk(clk),
              .rst(rst),
              .start(start),
              .dividend(dividend),
              .divisor(divisor),
              .quotient(quotient),
              .remainder(remainder),
              .done(done)
            );
  initial
```

```
  begin
    clk = 0;
    forever
      #10 clk = ~clk;
  end
  initial
  begin
    rst = 1;
    start = 0;
    dividend = 8'b0;
    divisor = 8'b0;
    #20;
    rst = 0;
    dividend = 8'd20;
    divisor = 8'd4;
    start = 1;
    #20 start = 0;
    wait(done);
    #10;
    $display("Test Case 1: 20 / 4 = %d R %d", quotient, remainder);
    if (quotient != 5 || remainder != 0)
      $display("Error: Expected Quotient = 5, Remainder = 0");
    dividend = 8'd25;
    divisor = 8'd4;
    start = 1;
    #20 start = 0;
    wait(done);
    #10;
    $display("Test Case 2: 25 / 4 = %d R %d", quotient, remainder);
    if (quotient != 6 || remainder != 1)
      $display("Error: Expected Quotient = 6, Remainder = 1");
    dividend = 8'd5;
    divisor = 8'd10;
    start = 1;
    #20 start = 0;
    wait(done);
    #10;
    $display("Test Case 3: 5 / 10 = %d R %d", quotient, remainder);
    if (quotient != 0 || remainder != 5)
      $display("Error: Expected Quotient = 0, Remainder = 5");
    dividend = 8'd15;
    divisor = 8'd1;
```

```
    start = 1;
    #20 start = 0;
    wait(done);
    #10;
    $display("Test Case 4: 15 / 1 = %d R %d", quotient, remainder);
    if (quotient != 15 || remainder != 0)
      $display("Error: Expected Quotient = 15, Remainder = 0");
    $finish;
  end
endmodule
```

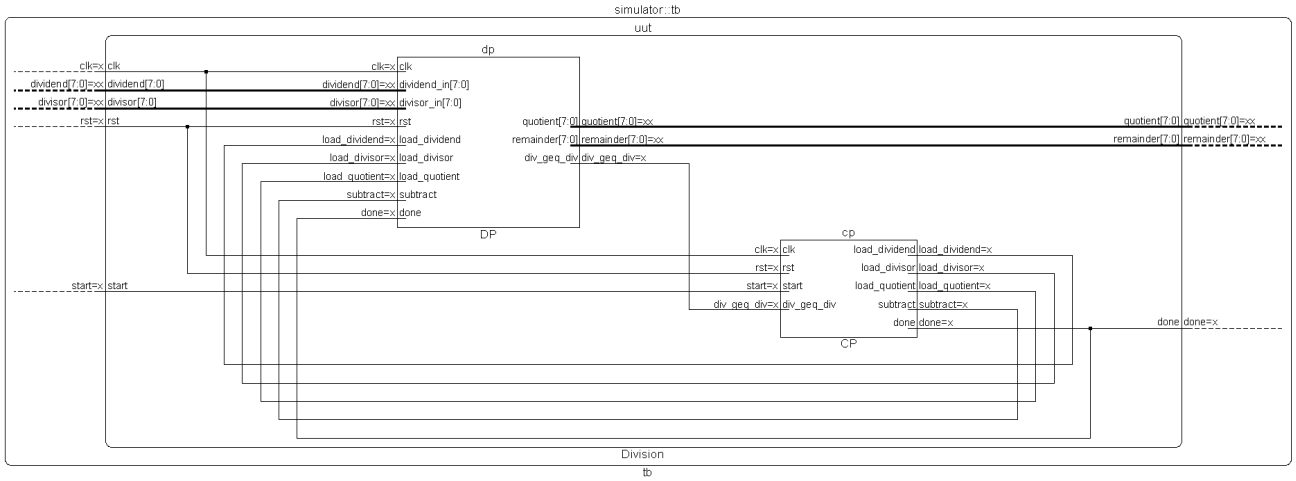
Waveform



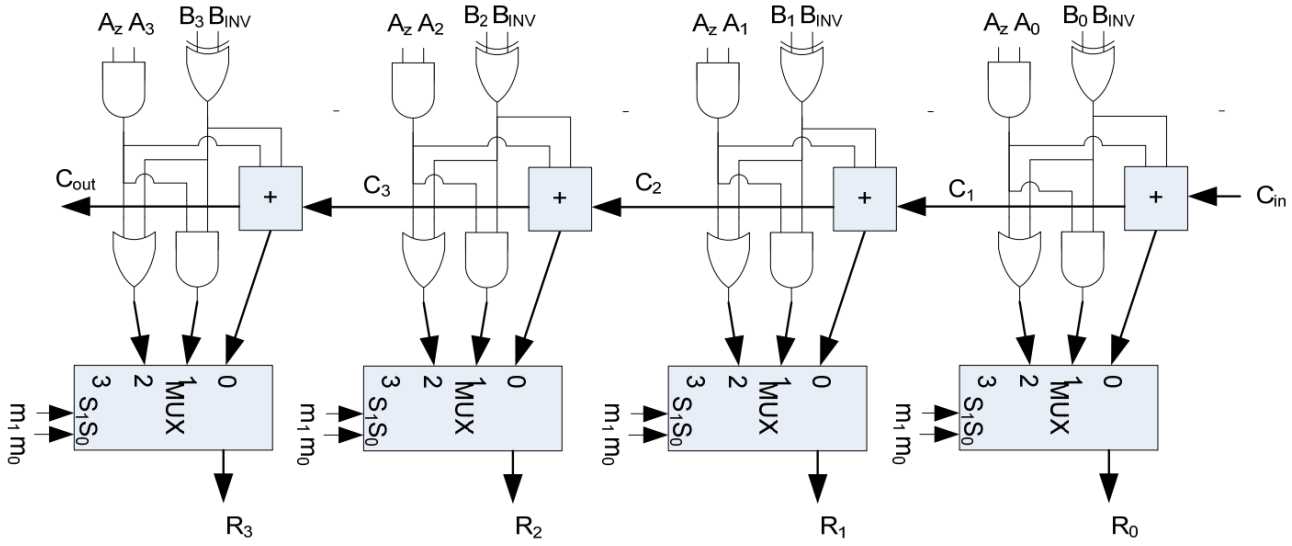
Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.clk tb.dividend tb.divisor tb.done tb.quotient tb.remainder tb.rst tb.start
run
Created default SHM database waves
ncsim> Created probe 1
ncsim> Test Case 1: 20 / 4 =   5 R   0
Test Case 2: 25 / 4 =   6 R   0
Error: Expected Quotient = 6, Remainder = 1
Test Case 3: 5 / 10 =   0 R   1
Error: Expected Quotient = 0, Remainder = 5
Test Case 4: 15 / 1 =  15 R   5
Error: Expected Quotient = 15, Remainder = 0
Simulation complete via $finish(1) at time 2660 NS + 0
./1qtb.v:58         $finish;\r
ncsim>
```

Schematic



2. A 4-bit ALU datapath is given below which can perform the following 5 operations: add, sub, AND, OR and negate B. Inputs are given by A={A3,A2,A1,A0}, B={B3,B2,B1,B0}, and Cin. Outputs are result R={R3, R2,R1,R0} and Cout. Numbers are represented in 2's complement form. Design and implement an optimized hardware for the datapath.



Source Code

```
module alu_fsm(reset, a, b, cin, out, sel, cout);
  input reset;
  input [3:0] a, b;
  input cin;
  input [2:0] sel;
  output reg [4:0] out;
  output cout;
  parameter S1 = 3'b000;
  parameter S2 = 3'b001;
  parameter S3 = 3'b010;
  parameter S4 = 3'b011;
  parameter S5 = 3'b100;
  parameter S6 = 3'b101;
  reg [2:0] c_state, n_state;
  reg r1, r2, r3, r4;
  always @(*)
  begin
    if (reset)
      c_state <= S6;
    else
```

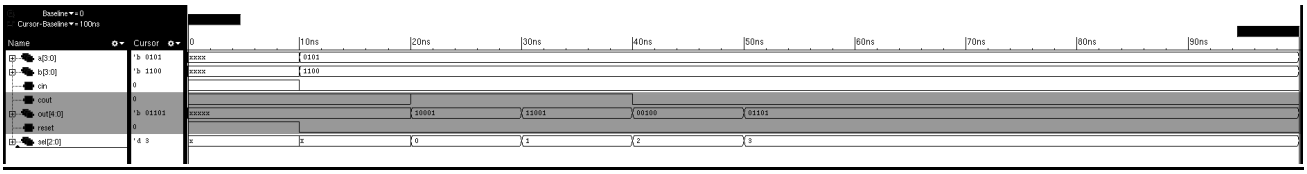
```
      c_state <= n_state;
  end
  always @(*)
  begin
    case (sel)
      3'b000:
        n_state <= S1;
      3'b001:
        n_state <= S2;
      3'b010:
        n_state <= S3;
      3'b011:
        n_state <= S4;
      3'b100:
        n_state <= S5;
      default:
        n_state <= S6;
    endcase
  end
  always @(*)
  begin
    case (c_state)
      S1:
      begin
        out <= a+b;
      end
      S2:
      begin
        out <= a-b;
      end
      S3:
      begin
        out <= a&b;
      end
      S4:
      begin
        out <= a|b;
      end
      S5:
      begin
        out <= ~b+1;
      end
```

```
      S6:
      begin
        out <= 5'bxxxx;
      end
    endcase
  end
  assign cout=out[4];
endmodule
```

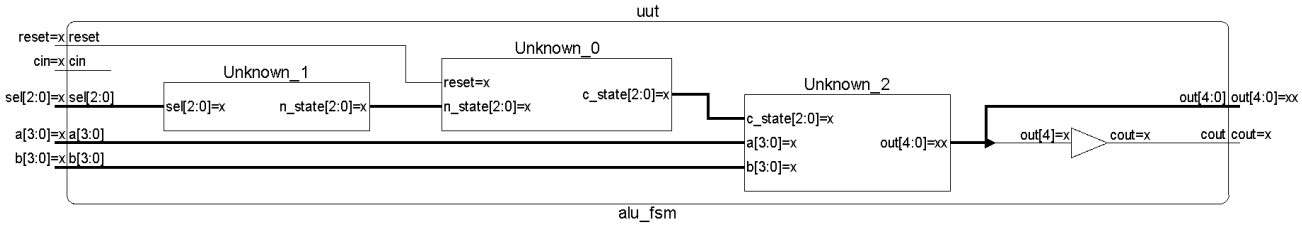
Test-bench

```
module tb();
  reg reset;
  reg [3:0] a, b;
  reg cin;
  reg [2:0] sel;
  wire [4:0] out;
  wire cout;
  alu_fsm uut(reset, a, b, cin, out, sel, cout);
  initial
  begin
    reset=1;
    #10 reset=0;
    a=4'b0101;
    b=4'b1100;
    cin=0;
    sel=2'bxx;
    #10 sel=2'b00;
    #10 sel=2'b01;
    #10 sel=2'b10;
    #10 sel=2'b11;
  end
  initial
  begin
    $display("reset=%d, a=%d, b=%d, cin=%d,
sel=%d,out=%d,cout=%d",reset,a,b,cin,sel,out,cout);
  end
  initial
  begin
    #100 $finish();
  end
endmodule
```

Waveform



Schematic



3. Design a datapath and control unit required to implement the function F= (a*b)/(c*d) using a single programmable (general purpose) datapath using ALU, any number of Registers, Multiplexers, buses etc.

Source Code

```
module datapath(out, sel, ldA, ldB, ldC, ldD, ldO, opsel, clk, a, b, c,
d);
  input sel, ldA, ldB, ldC, ldD, ldO, opsel, clk;
  output [31:0] out;
  input [15:0] a, b, c, d;
  wire [15:0] x, y, ra, rb;
  wire [31:0] rc, rd;
  wire [31:0] m, n, opout;
  MUX m1 (x, sel, a, c, clk);
  MUX m2 (y, sel, b, d, clk);
  MUX1 m3 (opout, opsel, m, n, clk);
  REG1 r1 (ra, ldA, x, clk);
  REG1 r2 (rb, ldB, y, clk);
  REG2 r3 (rc, ldC, opout, clk);
  REG2 r4 (rd, ldD, opout, clk);
  REG2 r5 (out, ldO, opout, clk);
  MUL alu0 (m, ra, rb);
  DIV alu1 (n, rc, rd);
endmodule
```

```
module MUX (dout, sel, in1, in2, clk);
  input [15:0] in1, in2;
  input sel, clk;
  output reg [15:0] dout;
  always@(posedge clk)
  begin
    if(sel == 0)
      dout <= in1;
    else
      dout <= in2;
  end
endmodule
module MUX1 (dout, sel, in1, in2, clk);
  input [31:0] in1, in2;
  input sel, clk;
  output reg [31:0] dout;
  always@(posedge clk)
  begin
    if(sel == 0)
      dout <= in1;
    else
      dout <= in2;
  end
endmodule
module REG1 (dout, ld, din, clk);
  input ld, clk;
  input [15:0] din;
  output reg [15:0] dout;
  always@(posedge clk)
  begin
    if(ld)
      dout <= din;
  end
endmodule
module REG2 (dout, ld, din, clk);
  input ld, clk;
  input [31:0] din;
  output reg [31:0] dout;
  always@(posedge clk)
  begin
    if(ld)
      dout <= din;
```

```
  end
endmodule
module MUL (dout, in1, in2);
  input [15:0]in1, in2;
  output reg [31:0] dout;
  always@(*)
  begin
    dout = in1 * in2;
  end
endmodule
module DIV (dout, in1, in2);
  input [31:0] in1, in2;
  output reg [31:0] dout;
  always@(*)
  begin
    dout = in1 / in2;
  end
endmodule
module controller (sel, ldA, ldB, ldC, ldD, ldO, opsel, done, start,
clk);
  input start, clk;
  output reg sel, ldA, ldB, ldC, ldD, ldO, opsel, done;
  reg [3:0] state;
  parameter s0 = 4'd0,
            s1 = 4'd1,
            s2 = 4'd2,
            s3 = 4'd3,
            s4 = 4'd4,
            s5 = 4'd5,
            s6 = 4'd6,
            s7 = 4'd7,
            s8 = 4'd8,
            s9 = 4'd9;

  always@(posedge clk)
  begin
    case(state)
      s0:
        if(start)
          state <= s1;
      s1:
        state <= s2;
```

```
      s2:
        state <= s3;
      s3:
        state <= s4;
      s4:
        state <= s5;
      s5:
        state <= s6;
      s6:
        #10 state <= s7;
      s7:
        state <= s8;
      s8:
        state <= s9;
      s9:
        state <= s9;
      default:
        state <= s0;
    endcase
  end
  always@(state)
  begin
    case(state)
      s0:
      begin
        #1 ldA = 0;
        ldB= 0;
        ldC = 0;
        ldD = 0;
        ldO = 0;
        opsel = 0;
        sel = 0;
      end
      s1:
      begin
        #1 sel = 0;
        ldA = 1;
        ldB = 1;
        opsel = 0;
      end
      s2:
      begin
```

```
        #1 opsel = 0;
        ldA = 0;
        ldB = 0;
      end
      s3:
      begin
        #1 ldC = 1;
      end
      s4:
      begin
        #1 ldC = 0;
        sel = 1;
        ldA = 1;
        ldB = 1;
      end
      s5:
      begin
        #1 opsel = 0;
      end
      s6:
      begin
        #1 ldD = 1;
      end
      s7:
      begin
        #1 opsel = 1;
        ldD = 0;
      end
      s8:
      begin
        #1 ldO = 1;
        opsel = 0;
      end
      s9:
      begin
        #1 done = 1;
        ldA = 0;
        ldB= 0;
        ldC = 0;
        ldD = 0;
        ldO = 0;
        opsel = 0;
```

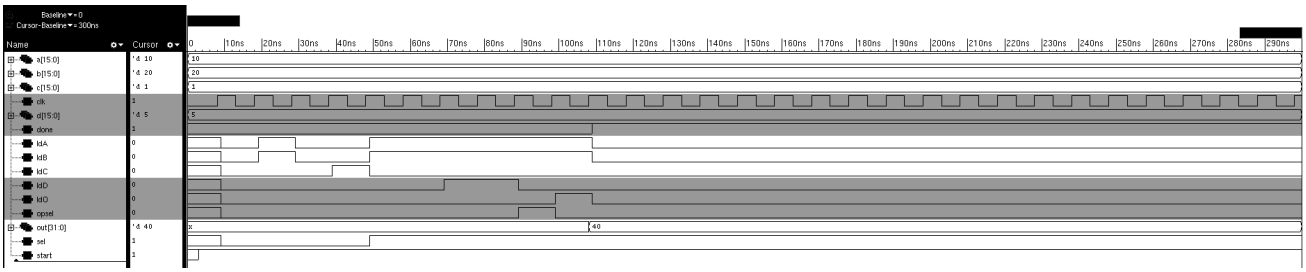
```
      end
      default:
      begin
        ldA = 0;
        ldB= 0;
        ldC = 0;
        ldD = 0;
        ldO = 0;
        opsel = 0;
        sel = 0;
        done = 0;
      end
    endcase
  end
endmodule
```

Test-bench

```
module tb();
  reg clk, start;
  wire sel, ldA, ldB, ldC, ldD, ldO, opsel, done;
  reg [15:0] a, b, c, d;
  wire [31:0] out;
  datapath DP (out, sel, ldA, ldB, ldC, ldD, ldO, opsel, clk, a, b, c,
d);
  controller CON (sel, ldA, ldB, ldC, ldD, ldO, opsel, done, start, clk);
  initial
  begin
    clk = 0;
    #3 start = 1;
    forever
      #5 clk = ~clk;
  end
  initial
  begin
    a = 16'd10;
    b = 16'd20;
    c = 16'd1;
    d = 16'd5;
    #300 $finish;
  end
```

```
  initial
  begin
    $monitor("Time: %0d, Output: %d, Done: %b", $time, out, done);
  end
endmodule
```

Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.a tb.b tb.c tb.clk tb.d tb.done tb.ldA tb.ldB tb.ldC tb.ldD tb.ldO tb.opsel tb.out tb.sel tb.start
run
Created default SHM database waves
ncsim> Created probe 1
ncsim> Time: 0, Output:          x, Done: x
Time: 108, Output:         40, Done: x
Time: 109, Output:         40, Done: 1
Simulation complete via $finish(1) at time 300 NS + 0
./3qtb.v:21     #300 $finish;
ncsim>
```

Schematic