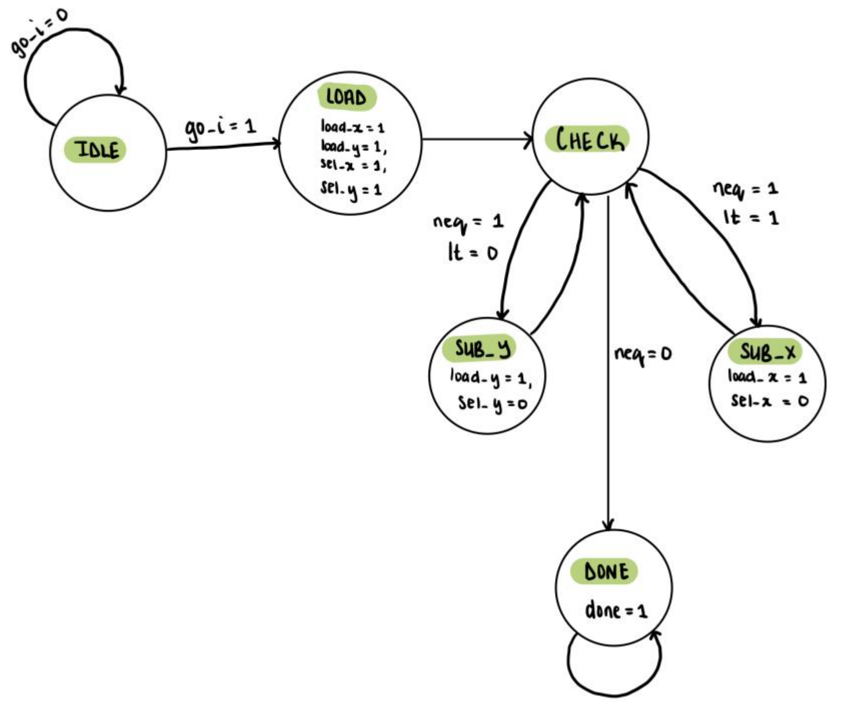
CDA 4203L
Computer System Design Lab

Lab 5 Report
GCD FSM and Datapath

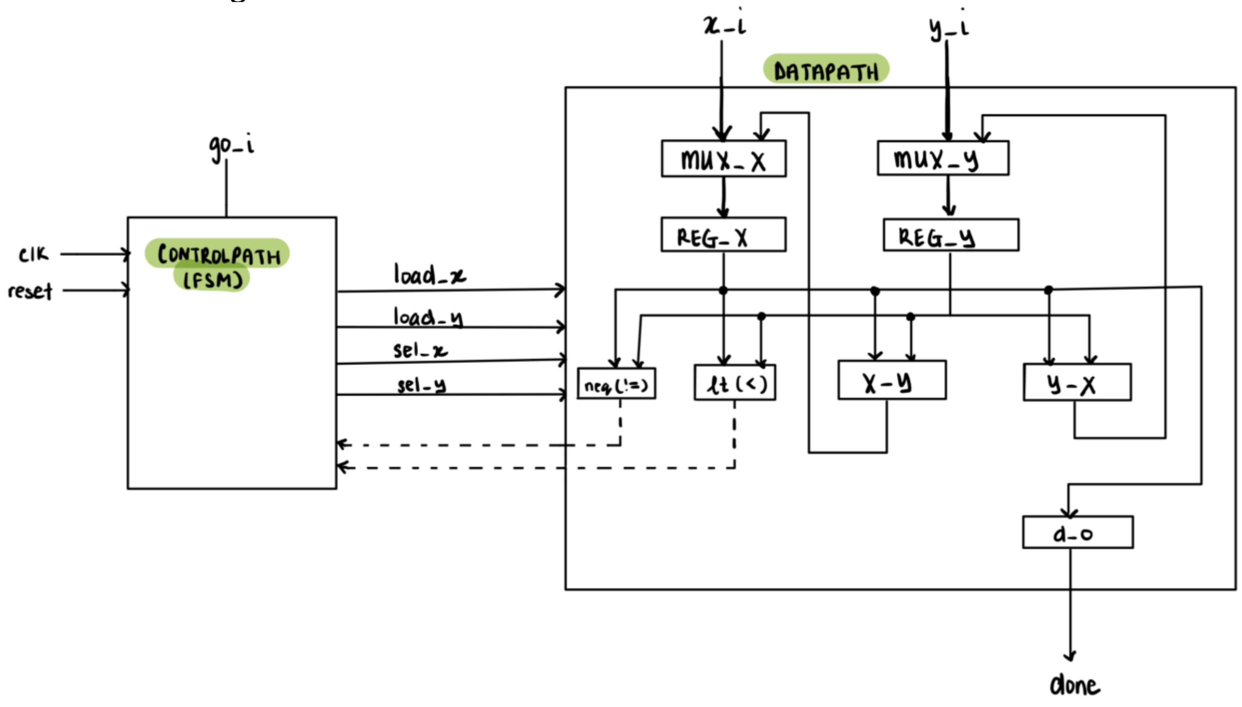
Today's Date:	4/4/2025
Team Members:	Brielle Ashmeade
	Claude Watson
	Tanisha Dutta
Work Distribution:	Briefly explain the tasks completed by each team member Question 1 – Brielle, Tanisha Question 2 - Claude
No. of Hours Spent:	5+
Exercise Difficulty: (Easy, Average, Hard)	Average
Any Other Feedback:	N/A

1. Show the state diagram of your GCD controller and behavioral components in the data path. Draw the FSMD and show the inputs/outputs. Briefly explain how the design works. *Use as many pages as needed.*

State Diagram:



FSMD Block Diagram:



<u>Brief Discussion of the Design</u>:

This design calculates the Greatest Common Divisor (GCD) of two positive integers using the Euclidean Algorithm. The system is composed of two main parts -

1. FSM Controller – Handles the control logic of the program
2. Datapath – Performs arithmetic operations

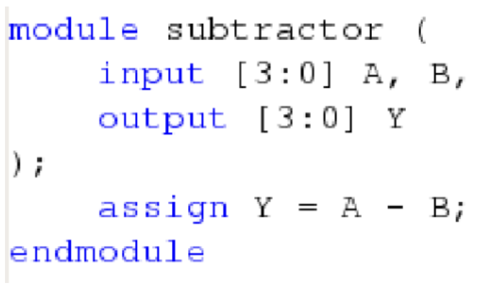
Step-by-step process of the design works:

1. Idle State: Controller waits for a start signal. Inputs x and y are stable during this time. Inputs x and y are stable during this time.
2. Load State: When start = 1, the finite state machines transitions to the Load state and loads inputs into registers X and Y
3. Check State: The finite state diagram checks the values of X and Y:
    a. If X > Y => Subtract Y from X (X = X – Y)
    b. If Y > X => Subtract X from Y (Y = Y – X)
    c. If X == Y => Move to the done state
4. Subtract State: Based on the comparison of X and Y, the appropriate subtraction is performed, and the result is loaded back into the correct register. The FSM then goes back into the Compare state.
5. Done State: When A == B, the GCD is found. The FSM moves into the done signal, and the result is the output of GCD.

2. Include the Verilog code of (a) data path components (Register (DFF), not-equal, less-than, subtractor, MUX, etc.); behavioral Verilog code of (b) FSM; (c) Your top level entity instantiating these two, (d) Testbench; and (e) Simulation waveforms. *Use as many pages as needed.*

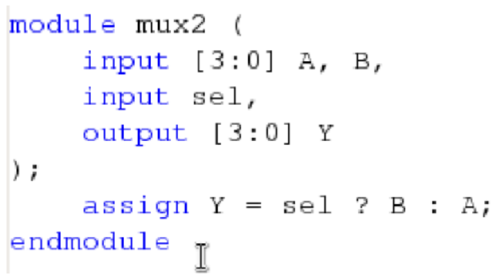
a). Datapath components:

i. Subtractor:



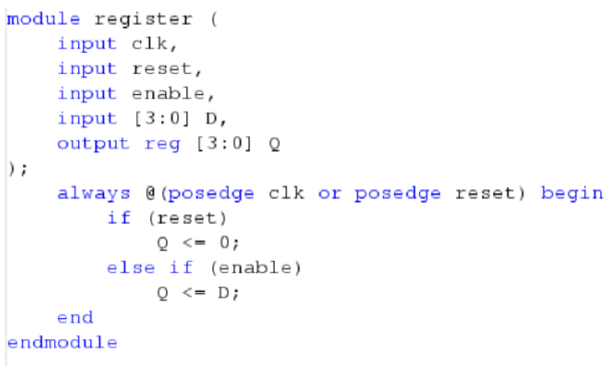
```
module subtractor (
    input [3:0] A, B,
    output [3:0] Y
);
    assign Y = A - B;
endmodule
```

ii. Mux2-1:



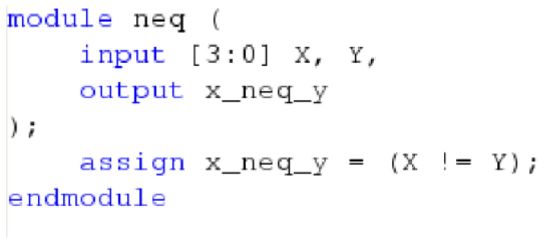
```
module mux2 (
    input [3:0] A, B,
    input sel,
    output [3:0] Y
);
    assign Y = sel ? B : A;
endmodule
```

iii. Register:



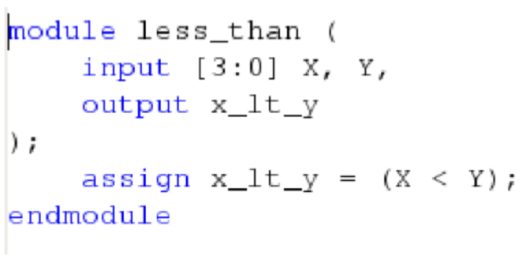
```
module register (
    input clk,
    input reset,
    input enable,
    input [3:0] D,
    output reg [3:0] Q
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            Q <= 0;
        else if (enable)
            Q <= D;
    end
endmodule
```

iv. Not Equal:



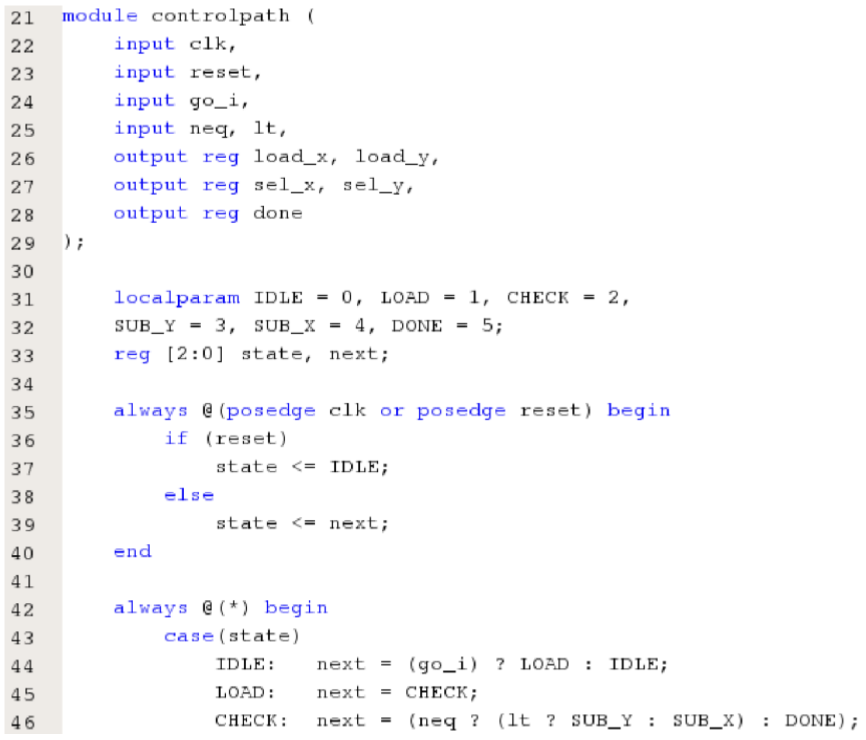
```
module neq (
    input [3:0] X, Y,
    output x_neq_y
);
    assign x_neq_y = (X != Y);
endmodule
```

v. Less Than:

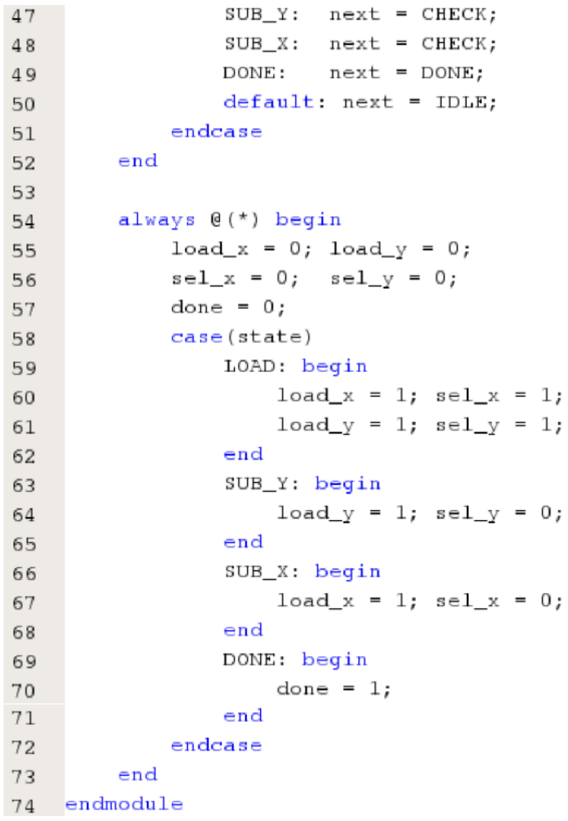


```
module less_than (
    input [3:0] X, Y,
    output x_lt_y
);
    assign x_lt_y = (X < Y);
endmodule
```

b). FSM:

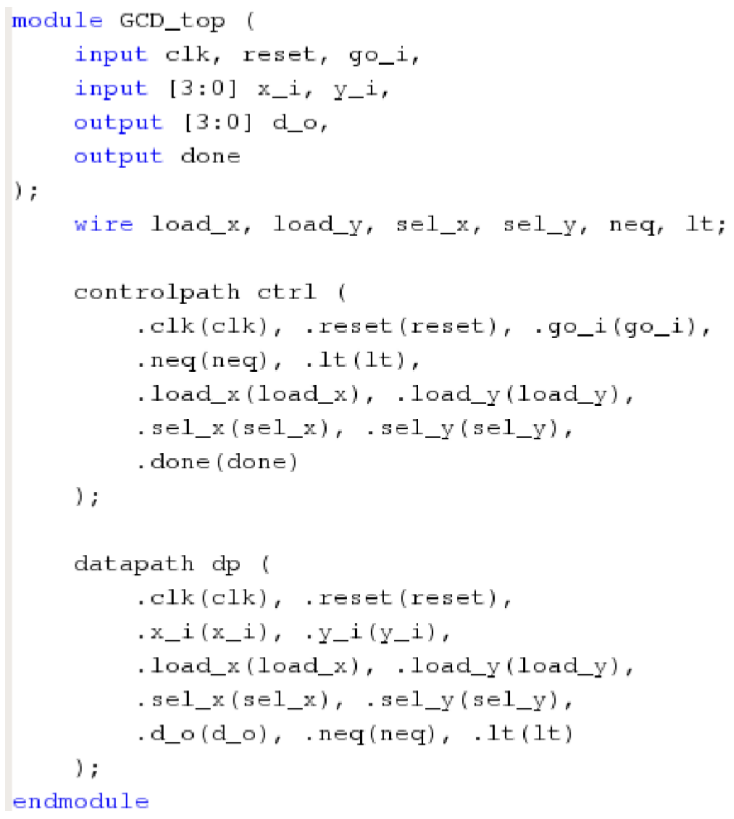


```
module controlpath (
    input clk,
    input reset,
    input go_i,
    input neq, lt,
    output reg load_x, load_y,
    output reg sel_x, sel_y,
    output reg done
);

    localparam IDLE = 0, LOAD = 1, CHECK = 2,
    SUB_Y = 3, SUB_X = 4, DONE = 5;
    reg [2:0] state, next;

    always @(posedge clk or posedge reset) begin
        if (reset)
            state <= IDLE;
        else
            state <= next;
    end

    always @(*) begin
        case(state)
            IDLE:   next = (go_i) ? LOAD : IDLE;
            LOAD:   next = CHECK;
            CHECK:  next = (neq ? (lt ? SUB_Y : SUB_X) : DONE);
```



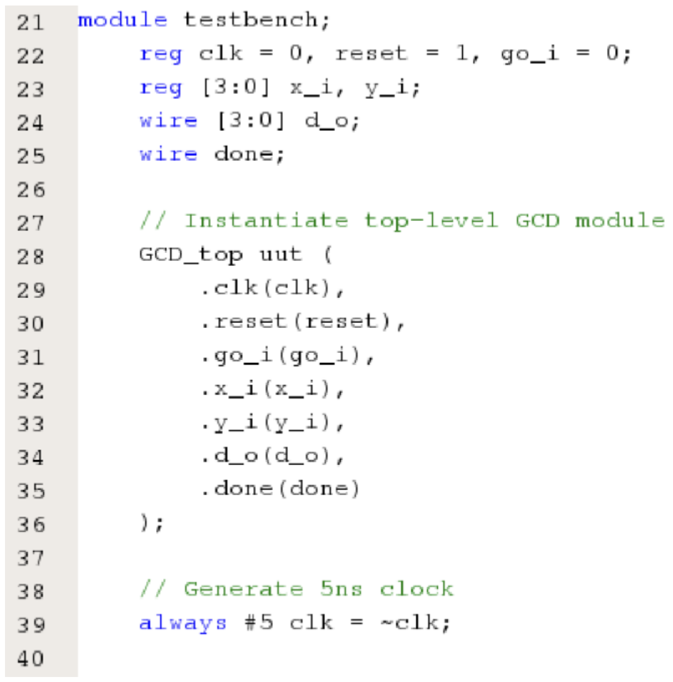
```verilog
            SUB_Y:  next = CHECK;
            SUB_X:  next = CHECK;
            DONE:   next = DONE;
            default: next = IDLE;
        endcase
    end

    always @(*) begin
        load_x = 0; load_y = 0;
        sel_x = 0;  sel_y = 0;
        done = 0;
        case(state)
            LOAD: begin
                load_x = 1; sel_x = 1;
                load_y = 1; sel_y = 1;
            end
            SUB_Y: begin
                load_y = 1; sel_y = 0;
            end
            SUB_X: begin
                load_x = 1; sel_x = 0;
            end
            DONE: begin
                done = 1;
            end
        endcase
    end
endmodule
```

c). Top-Level Entity:

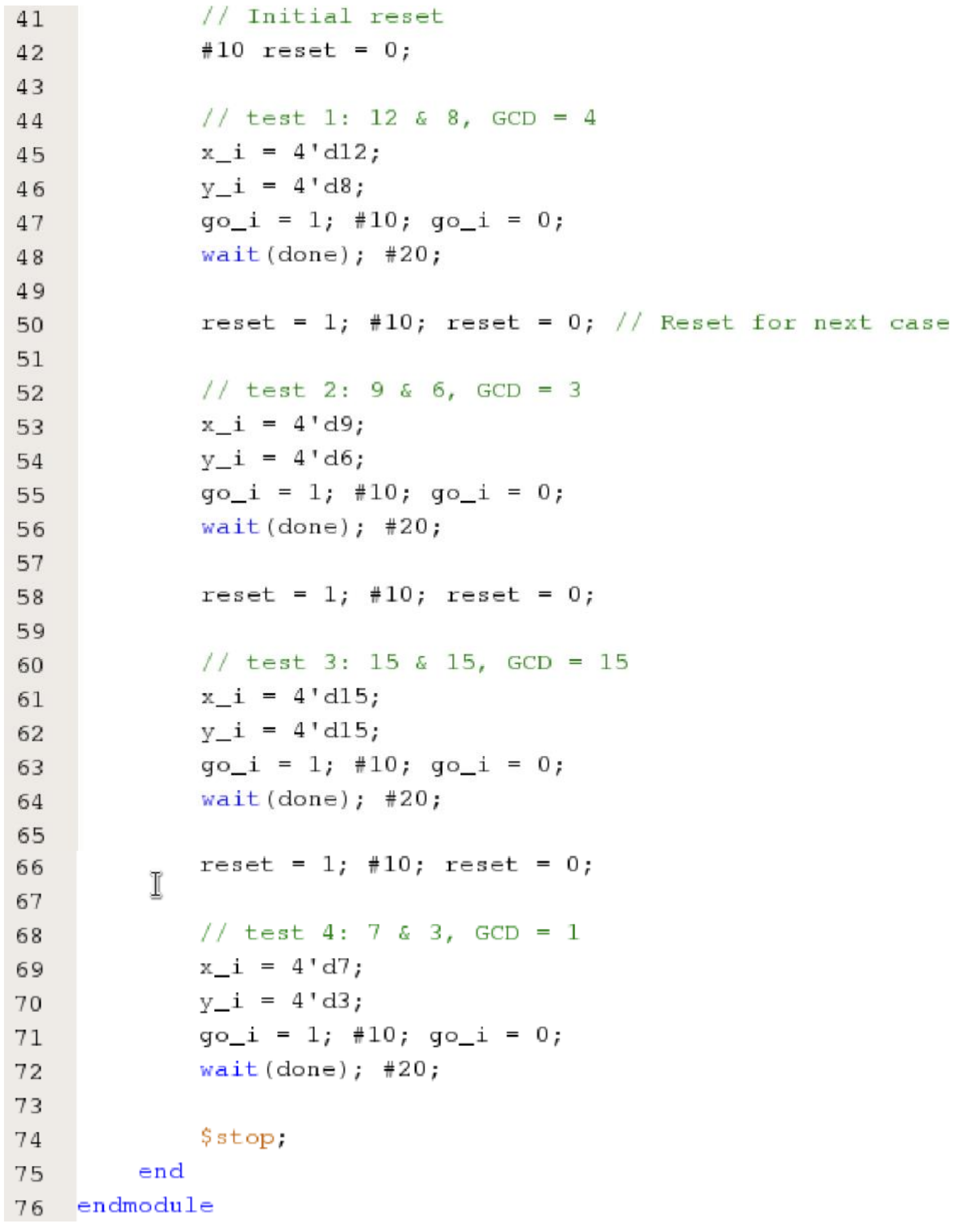


```verilog
module GCD_top (
    input clk, reset, go_i,
    input [3:0] x_i, y_i,
    output [3:0] d_o,
    output done
);
    wire load_x, load_y, sel_x, sel_y, neq, lt;

    controlpath ctrl (
        .clk(clk), .reset(reset), .go_i(go_i),
        .neq(neq), .lt(lt),
        .load_x(load_x), .load_y(load_y),
        .sel_x(sel_x), .sel_y(sel_y),
        .done(done)
    );

    datapath dp (
        .clk(clk), .reset(reset),
        .x_i(x_i), .y_i(y_i),
        .load_x(load_x), .load_y(load_y),
        .sel_x(sel_x), .sel_y(sel_y),
        .d_o(d_o), .neq(neq), .lt(lt)
    );
endmodule
```

d). Testbench:

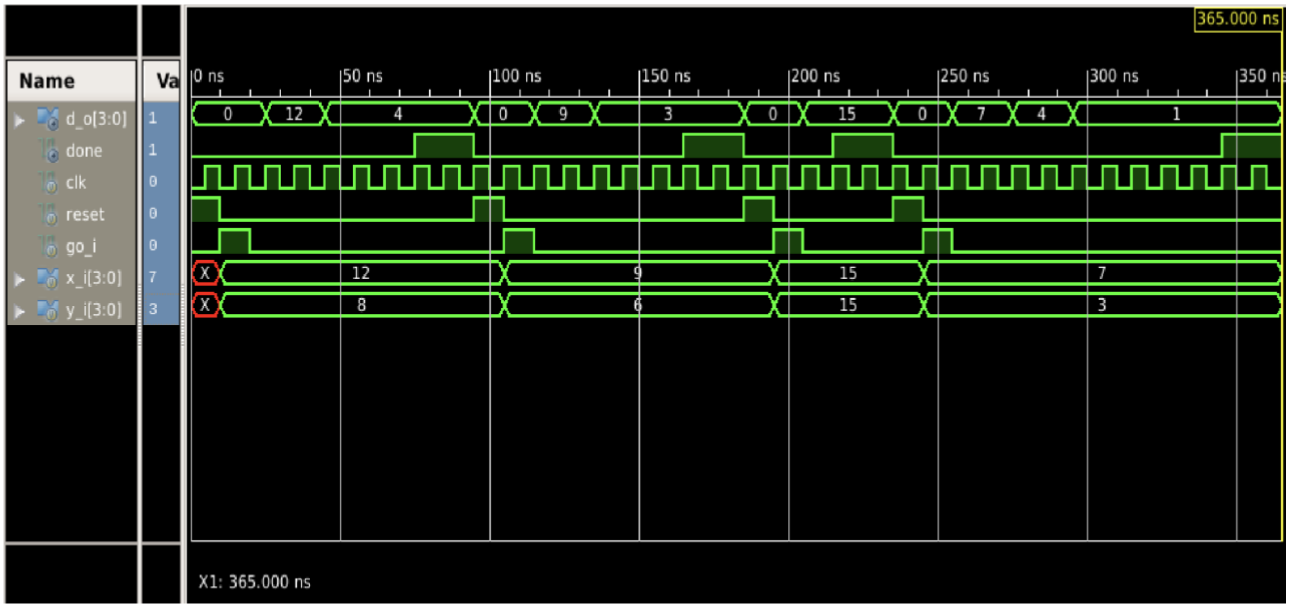


```verilog
module testbench;
    reg clk = 0, reset = 1, go_i = 0;
    reg [3:0] x_i, y_i;
    wire [3:0] d_o;
    wire done;

    // Instantiate top-level GCD module
    GCD_top uut (
        .clk(clk),
        .reset(reset),
        .go_i(go_i),
        .x_i(x_i),
        .y_i(y_i),
        .d_o(d_o),
        .done(done)
    );

    // Generate 5ns clock
    always #5 clk = ~clk;

```



```
        // Initial reset
        #10 reset = 0;

        // test 1: 12 & 8, GCD = 4
        x_i = 4'd12;
        y_i = 4'd8;
        go_i = 1; #10; go_i = 0;
        wait(done); #20;

        reset = 1; #10; reset = 0; // Reset for next case

        // test 2: 9 & 6, GCD = 3
        x_i = 4'd9;
        y_i = 4'd6;
        go_i = 1; #10; go_i = 0;
        wait(done); #20;

        reset = 1; #10; reset = 0;

        // test 3: 15 & 15, GCD = 15
        x_i = 4'd15;
        y_i = 4'd15;
        go_i = 1; #10; go_i = 0;
        wait(done); #20;

        reset = 1; #10; reset = 0;

        // test 4: 7 & 3, GCD = 1
        x_i = 4'd7;
        y_i = 4'd3;
        go_i = 1; #10; go_i = 0;
        wait(done); #20;

        $stop;
    end
endmodule
```

e). Simulation Waveform:


365.000 ns
Name
Va
d_o[3:0]
done
clk
reset
go_i
x_i[3:0]
y_i[3:0]
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
0
12
4
0
9
3
0
15
0
7
4
1
X
12
9
15
7
X
8
6
15
3
X1: 365.000 ns