Design and implement GCD (Greatest Common Divisor) computation for two given numbers.
Draw finite state machine and data path.
Write the Verilog code and include output waveforms.

Contents

1 AIM

Design and implement GCD (Greatest Common Divisor) computation for two given numbers. Draw finite state machine and data path. Write the Verilog code and include output waveforms.

2 Objective

a) Design a GCD (Greatest Common Divisor) computation for two 8-bit numbers (x,y) and verify its functionalityin Xilinx Vivado.
b) Perform synthesis of the design and evaluate LUT's, FFs, Power and Delay of the GCD.

3 Theory

3.1 Greatest Common divisor (GCD)

The GCD (Greatest Common Divisor) of two numbers is the largest positive integer that divides both numbers without leaving a remainder. There are many ways to find the GCD of two numbers, including the Subtraction Method.The Subtraction Method is a simple and iterative algorithm for finding the GCD of two numbers. The basic idea is to subtract the smaller number from the larger number until both numbers become equal. The resulting number is the GCD of the original two numbers. Here is the step-by-step process for finding the GCD of two numbers using the Subtraction Method:

Take two numbers "a" and "b", where "a" is greater than or equal to "b". Subtract "b" from "a". If the result is greater than or equal to "b", replace "a" with the result and go back to step 2. Otherwise, let "a" be the smaller number and let "b" be the difference.The resulting number is the GCD of the original two numbers.
For example, let's find the GCD of 42 and 28 using the Subtraction Method:

a = 42 and b = 28
Subtract "b" from "a": 42 - 28 = 14. Since 14 is less than 28, let "a = 28" and "b = 14".
Subtract "b" from "a": 28 - 14 = 14. Since both numbers are equal, stop.
The GCD of 42 and 28 is 14.
The Subtraction Method is a simple and easy-to-understand algorithm for finding the GCD of two numbers. However, it can be inefficient for large numbers or when the numbers are relatively prime (have no common factors). Other algorithms, such as the Euclidean Algorithm, are more efficient for these cases.

4 Verilog Code

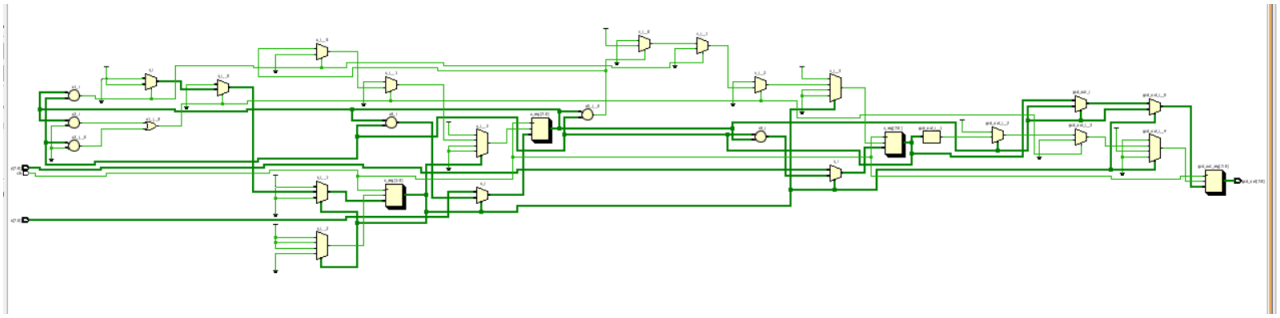
4.1 Code

```
`timescale 1ns / 1ps
module gcd_n(
input wire [7:0]x,
input wire [7:0]y,
input clk,
output reg [7:0]gcd_out);


reg [7:0]v;
reg [7:0]u;
reg [1:0]s=0;


always@(posedge clk)
begin
    case(s)
    2'b00: begin
                u=x;
                v=y;
                s=2'b01;
           end

     2'b01:begin
           if(u==0||v==0)
           begin
                    if(u==0)
                     gcd_out=v;
                    else if(v==0)
                     gcd_out=u;
                     s=2'b00;
           end
           else
           begin
                 if(u!=v)
                      if(u>v)
                      if(u>v)
                      begin
                       u=u-v;
                       s=2'b01;
                       end
                      else
                      begin
                       v=v-u;
                       s=2'b01;
                      end
                 else
                      s=2'b11;
           end
           end
        2'b11:
        begin
        gcd_out=u;
        s=2'b00;
        end
        endcase
end
endmodule
```

4.2 Test Bench

```
`timescale 1ns / 1ps
module gcd_n_tb();


reg [7:0]a,b;
reg clk;
wire [7:0]c;

gcd_n g(a,b,clk,c);

initial
begin
clk=1'b0;

forever #5 clk=~clk;
end
//in=1'b0;
initial
begin
 a=8'd4;b=8'd0;
#100
a=8'd128;b=8'd64;
#100
a=8'd22;b=8'd33;
#100
a=8'd45;b=8'd81;
#200
$finish;

end
endmodule
```

4.3 RTL schemetic



4.4 Functionality

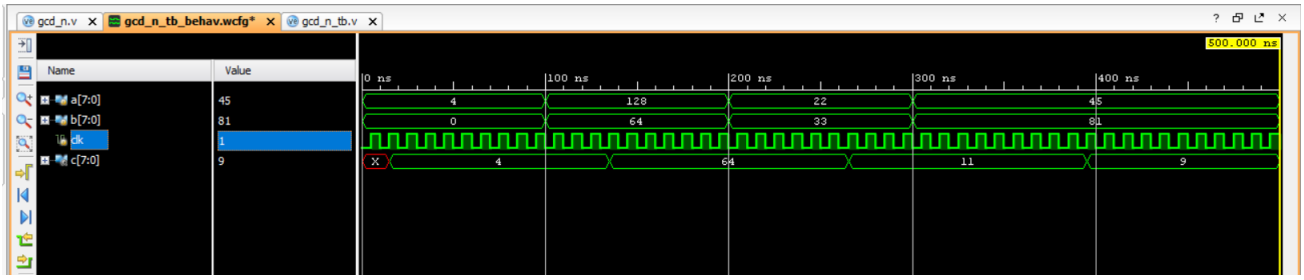


Figure 1: Output Waveform

4.5 LUTs,Flip-Flops,Power,Delays Observations

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIe %	Start	Elapsed	Strategy
synth_1	constrs_1	synth_design Complete!								49	26	0	0	0.000	9/27/23 6:02 PM	00:00:36	Vivado Synthesis Defaul
impl_1	constrs_1	Not started															Vivado Implementation D

Figure 2: Number of LUT's

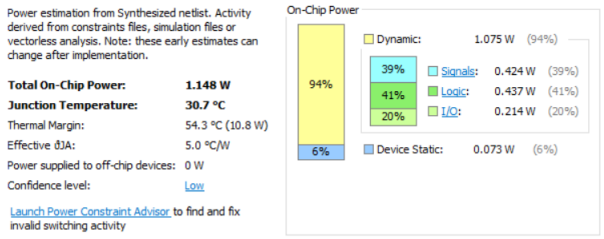
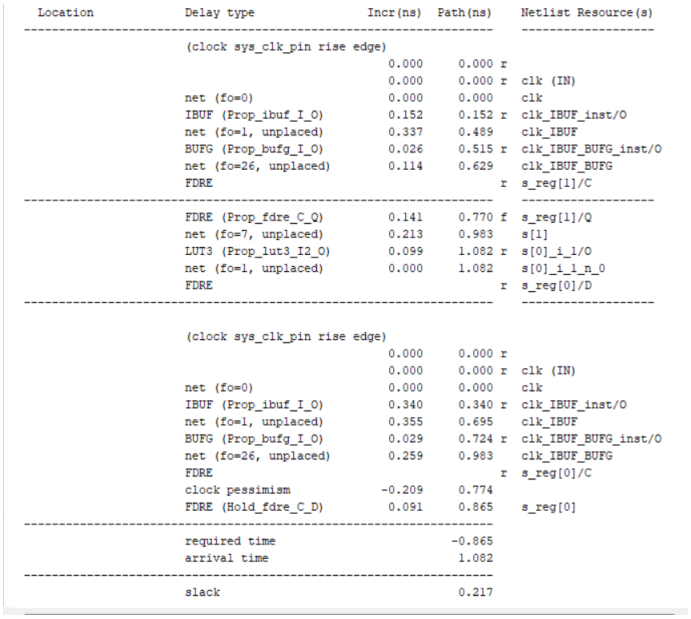


Figure 3: Power



Location	Delay type	Incr(ns)	Path(ns)		Netlist Resource(s)
	(clock sys_clk_pin rise edge)				
		0.000	0.000	r	
		0.000	0.000	r	clk (IN)
	net (fo=0)	0.000	0.000		clk
	IBUF (Prop_ibuf_I_O)	0.152	0.152	r	clk_IBUF_inst/O
	net (fo=1, unplaced)	0.337	0.489		clk_IBUF
	BUFG (Prop_bufg_I_O)	0.026	0.515	r	clk_IBUF_BUFG_inst/O
	net (fo=26, unplaced)	0.114	0.629		clk_IBUF_BUFG
	FDRE			r	s_reg[1]/C
	FDRE (Prop_fdre_C_Q)	0.141	0.770	f	s_reg[1]/Q
	net (fo=7, unplaced)	0.213	0.983		s[1]
	LUT3 (Prop_lut3_I2_O)	0.099	1.082	r	s[0]_i_1/O
	net (fo=1, unplaced)	0.000	1.082		s[0]_i_1_n_0
	FDRE			r	s_reg[0]/D
	(clock sys_clk_pin rise edge)				
		0.000	0.000	r	
		0.000	0.000	r	clk (IN)
	net (fo=0)	0.000	0.000		clk
	IBUF (Prop_ibuf_I_O)	0.340	0.340	r	clk_IBUF_inst/O
	net (fo=1, unplaced)	0.355	0.695		clk_IBUF
	BUFG (Prop_bufg_I_O)	0.029	0.724	r	clk_IBUF_BUFG_inst/O
	net (fo=26, unplaced)	0.259	0.983		clk_IBUF_BUFG
	FDRE			r	s_reg[0]/C
	clock pessimism	-0.209	0.774		
	FDRE (Hold_fdre_C_D)	0.091	0.865		s_reg[0]
	required time		-0.865		
	arrival time		1.082		
	slack		0.217		

Figure 4: Delay

5 Observation

Circuit parameters of GCD

Circuit	LUT's	flip flops	Delay(ns)	power(W)
GCD	49	26	1.082	1.075

6 Results

I Sucesfully Calculated the no:of LUT's, flip flops , delay and power required for a GCD (Greatest Common Divisor) computation for two numbers which are noted above in table.