# 

# EE424 DIGITAL SYSTEM DESING WITH VERILOG TERM PROJECT REPORT

## MEMORY SORTING AND GCD ALGORITHM

**KERİM TURAK 17050211011**

## **Inroduction**

In this project, it is aimed to apply the BRAM algorithm and to sort the data in this ram from largest to smallest or vice versa with a second module, and in the last step to find the greatest common divisor of the two largest numbers in this BRAM with the GCD module using the GCD algorithm. The following steps were followed to finish the project.

* **DUAL PORT BRAM MODULE DESING**
* **SORT MODULE DESIGN**
* **GCD MODULE DESIGN**

# **DUAL PORT BRAM MODULE DESING**

I thought it would be easier and more logical to use a dual port ram module. As a result of my research, I found the code on page 109 of the document of Vivado Design Suite User Guide suitable for this purpose and verified it by writing the testbench. And code is as below. **Code Source**: [Link](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug901-vivado-synthesis.pdf)

`timescale 1ns / 1ps

module memory(clka,clkb,ena,enb,wea,web,addra,addrb,mem\_inA,mem\_inB,mem\_outA,mem\_outB);

input clka,clkb,ena,enb,wea,web;

input [3:0] addra,addrb;

input signed[7:0] mem\_inA,mem\_inB;

output signed[7:0] mem\_outA,mem\_outB;

reg[7:0] ram [15:0];

reg[7:0] mem\_outA,mem\_outB;

always @(posedge clka) begin if (ena)

begin

if (wea)

ram[addra] <= mem\_inA;

mem\_outA <= ram[addra];

end

end

always @(posedge clkb) begin if (enb)

begin

if (web)

ram[addrb] <= mem\_inB;

mem\_outB <= ram[addrb];

end

end

endmodule

Here, I used port 1 initially to enter random values into BRAM. The other port was used by SORT and GCD module.

# **SORT MODULE**

I was able to sort the memory using the codes I wrote before, but I used a double for loop there and these codes gave an error during synthesis.

`timescale 1ns / 1ps

module sort(clk, sort\_in0, sort\_in1, sort\_out0, sort\_out1, sort, done, state);

input clk;

input state;

output done;

input signed [7:0]sort\_in0, sort\_in1;

output reg signed [7:0] sort\_out0, sort\_out1;

input sort;

reg signed [7:0] temp [1:0];

reg sign\_of\_difference;

reg [7:0] difference;

always@(posedge clk) begin

if(sort) begin

temp[0] <= sort\_in0;

temp[1] <= sort\_in1;

difference <= temp[1]-temp[0];

sign\_of\_difference <= difference[7];

if (sign\_of\_difference)begin

sort\_out0 <= temp[1];

sort\_out1 <= temp[0];

end

else begin

sort\_out0 <= temp[0];

sort\_out1 <= temp[1];

end

end

end

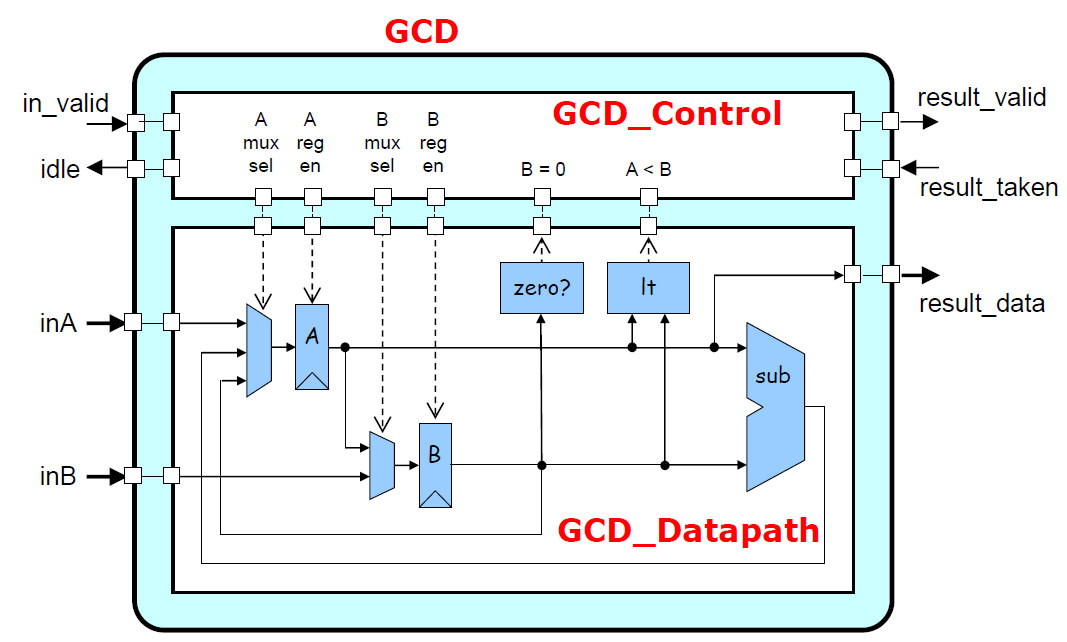
assign done = sort && state ? 1:0;

endmodule

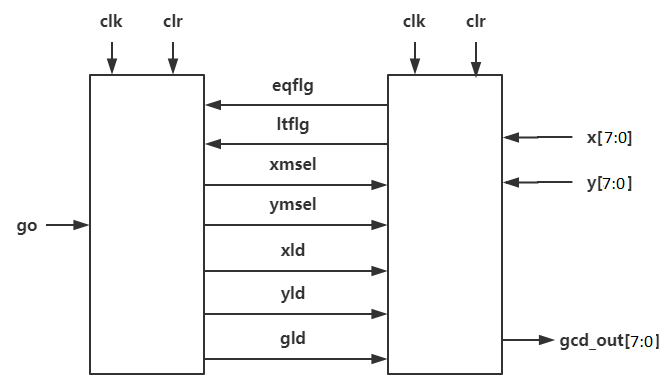
Later, as a result of my research, I coded a 3-line code block about ordering 2 numbers on stackoverflow in a way suitable for my own module. The two values at the input were recorded in a register and their difference was taken, and the last sign bit of their difference was recorded in a separate register and the numbers could be sorted with the help of a single if else block according to the sign of the operation result, that is, whether it was 1 or zero. Related stackoverflow discussion thread: [Link](related%20stackoverflow%20discussion%20thread) . As a result, I created the code below, connected it to RAM and verified it on testbench.Here, the always block at the bottom sets the done variable to 1 if the data is sorted when the sort signal arrives, if the state returns as 1. State indicates that the sorting process is finished here.

# **GCD MODULE**

As a result of my research to design the GCD module, I found resources in line with the course materials.



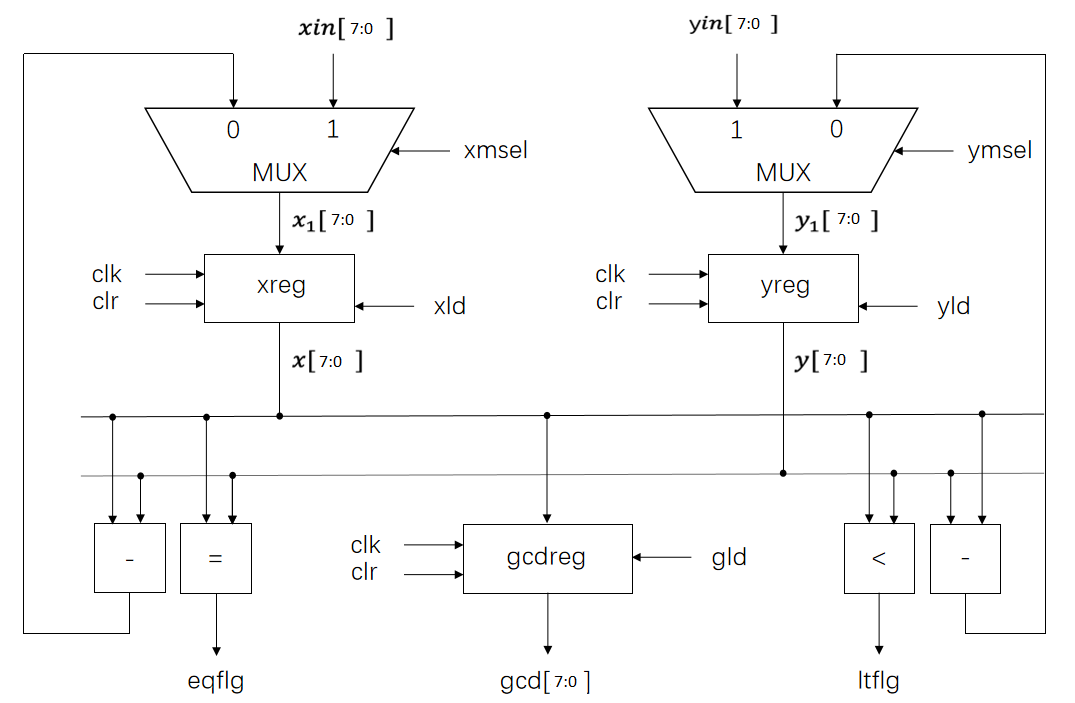
As seen in the picture above, we need to design 2 submodules to design the GCD module. The first is the datapath module and the second is the control module.

And I made the following design based on the picture above.

Datapath

Control

For Datapath, the design in the figure was drawn similar to the below one.



The Datapath module was written in line with this schematic as follows.

//……………

always @(posedge clk) begin //gcdreg

if(clr == 1) begin

gcd <= 0;

end

else if(gld == 1) begin

gcd <= x;

end

else begin

gcd <= gcd;

end

end

assign xmy = x - y;

assign ymx = y - x;

assign eqflg = x == y ? 1:0;

assign ltflg = x < y ? 1:0;

endmodule

`timescale 1ns / 1ps

module gcd\_datapath(

input clk,

input clr,

input xmsel,

input ymsel,

input xld,

input yld,

input gld,

input signed [7:0] xin,

input signed [7:0] yin,

output eqflg,

output ltflg,

output /\*wire\*/ reg [7:0] gcd);

wire signed [7:0] xmy, ymx;

wire signed [7:0] x1, y1;

reg signed [7:0] x, y;

assign x1 = xmsel ? xin:xmy;

assign y1 = ymsel ? yin:ymx;

always @(posedge clk) begin //xreg

if(clr == 1) begin

x <= 0;

end

else if(xld == 1) begin

x <= x1;

end

else begin

x <= x;

end

end

always @(posedge clk) begin //yreg

if(clr == 1) begin

y <= 0;

end

else if(yld == 1) begin

y <= y1;

end

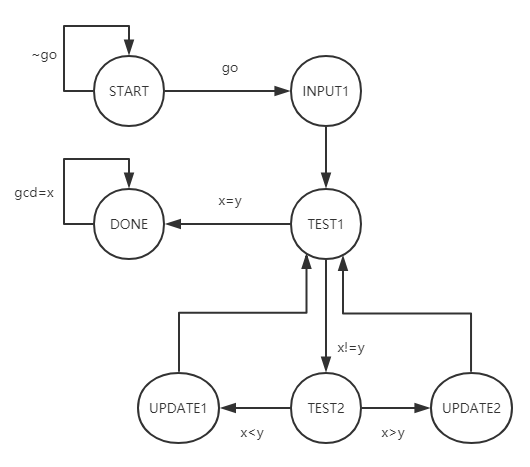
else begin

y <= y;

end

end

//…………………

For the control module, the states were determined as follows and the codes were written appropriately.

F

`timescale 1ns / 1ps

module gcd\_control(

input clk,

input clr,

input eqflg,

input ltflg,

output reg stop,

output reg xld,

output reg yld,

output reg xmsel,

output reg ymsel,

output reg gld);

reg [2:0] current\_state, next\_state;

parameter start = 3'b000,

input1 = 3'b001,

test1 = 3'b010,

test2 = 3'b011,

update1 = 3'b100,

update2 = 3'b101,

done = 3'b110;

always @(posedge clk) begin

if(clr) begin

current\_state <= start;

stop <= 0;

end

else

current\_state <= next\_state;

end

always @(\*) begin

case(current\_state)

start:

next\_state <= input1;

input1: begin

next\_state <= test1;

xmsel <= 1;

ymsel <= 1;

xld <= 1;

yld <= 1;

end

test1: begin

xmsel <= 0;

ymsel <= 0;

xld <= 0;

yld <= 0;

if(eqflg == 1)

next\_state <= done;

else

next\_state <= test2;

end

test2:begin

xmsel <= 0;

ymsel <= 0;

xld <= 0;

yld <= 0;

if(eqflg == 1)

next\_state <= done;

else if(ltflg == 1)

next\_state <= update1;

else

next\_state <= update2;

end

update1: begin

next\_state <= test1;

yld <= 1;

xld <= 0;

ymsel <= 0;

end

update2:begin

xld <= 1;

yld <= 0;

xmsel <= 0;

next\_state <= test1;

end

done: begin

next\_state <= done;

gld <= 1;

stop <= 1;

end

default:

next\_state = start;

endcase

end

endmodule

My GCD module is as following.

`timescale 1ns / 1ps

module gcd(gcd\_out,STOP,clr,clka,clkb,ena,enb,wea,web,addra,addrb,da,mem\_outA,mem\_outB,sort\_out0, sort\_out1,sort, done, state);

input clr;

output signed [7:0] gcd\_out;

output reg STOP;

wire xsel, ysel, xmld, ymld, gmld, eqmflg, ltmflg;

wire stop;

input signed[7:0] da;

input clka,clkb,ena,enb,wea,web;

input [3:0] addra,addrb;

wire signed[7:0] min\_a,min\_b;

output signed[7:0] mem\_outA, mem\_outB;

output signed [7:0] sort\_out0, sort\_out1;

input sort;

output done;

input state;

memory mem(.clka(clka), .clkb(clkb), .ena(ena), .enb(enb), .wea(wea),

.web(web), .addra(addra), .addrb(addrb), .mem\_inA(min\_a), .mem\_inB(sort\_out1),

.mem\_outA(mem\_outA), .mem\_outB(mem\_outB));

sort srt(.clk(clka), .sort\_in0(mem\_outA), .sort\_in1(mem\_outB) , .sort\_out0(sort\_out0) ,

.sort\_out1(sort\_out1), .sort(sort), .done(done), .state(state));

assign min\_a = sort ? sort\_out0:da;

assign dia = sort ? sort\_out0:da; //mux

gcd\_datapath U1(.clk(clka), .clr(clr), .xmsel(xsel), .ymsel(ysel), .xld(xmld),

.yld(ymld), .gld(gmld), .xin(mem\_outA), .yin(mem\_outB), .eqflg(eqmflg),

.ltflg(ltmflg), .gcd(gcd\_out));

gcd\_control U2(.clk(clka), .clr(clr), .eqflg(eqmflg), .ltflg(ltmflg), .stop(stop), .xld(xmld),

.yld(ymld), .xmsel(xsel), .ymsel(ysel), .gld(gmld));

always @ (posedge clka) begin

STOP <= stop;

end

endmodule

My GCD testbench module is as following.

`timescale 1ns / 1ps

module gcdtest();

reg clr;

wire signed[7:0] gcd\_out;

wire stop;

reg clka, clkb, ena, enb, wea, web;

reg [3:0] addra, addrb;

reg [7:0] da;

wire [7:0] mou\_a, mou\_b;

wire [7:0] sou0, sou1;

reg sort;

integer i;

wire done;

reg state;

wire [7:0] Y;

gcd uut(gcd\_out,STOP,

clr,clka,clkb,

ena,enb,wea,web,

addra,addrb,da,

mou\_a,mou\_b,

sou0, sou1,

sort, done, state);

initial

begin

i = 0;state =0; clr = 1;

clka = 1; clkb = 1; ena = 0; enb = 0; wea= 0; web = 0; addra = 0; addrb =1;

ena = 1; wea = 1; sort = 0;

addra = 0; da = -8; #10;

enb = 1; addrb = 0;

addra = 1; da = -6; #10; addra = 2; da = -7; #10; addra = 3; da = -4; #10;

addra = 4; da = -5; #10; addra = 5; da = -2; #10; addra = 6; da = -3; #10;

addra = 7; da = 0; #10; addra = 8; da = -1; #10; addra = 9; da = 2; #10;

addra = 10; da = 1; #10; addra = 11; da = 4; #10; addra = 12; da = 3; #10;

addra = 13; da = 6; #10; addra = 14; da = 10; #10; addra = 15; da = 15; #10;

sort=1; ena = 1; enb = 1; ena = 1; enb = 1;

for(i=0; i<15; i = i+1)begin

wea = 0; web = 0; addra = i; addrb =i+1; #30;

wea = 1; web = 1; addra = i; addrb =i+1; #30;

end

state = 1;

end

always #5 clka = ~clka;

always #5 clkb = ~clkb;

initial begin

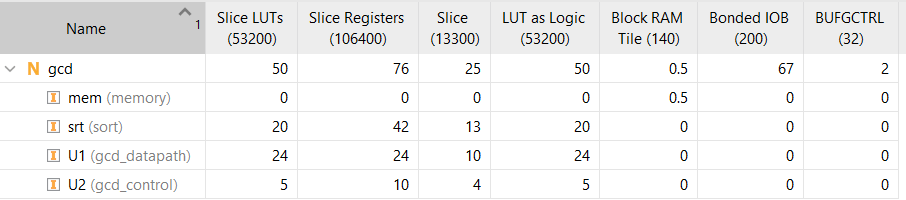
#1300;

clr = 0;

end

endmodule

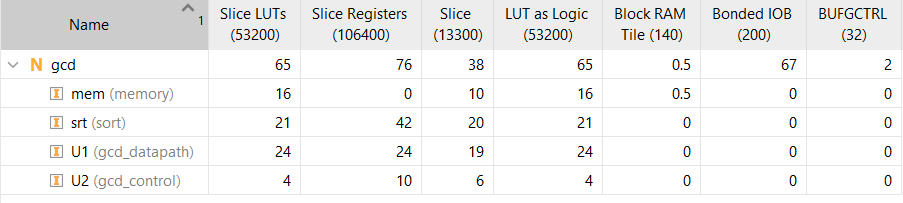
FOR 100 MHZ



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

FOR 250 MHZ



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

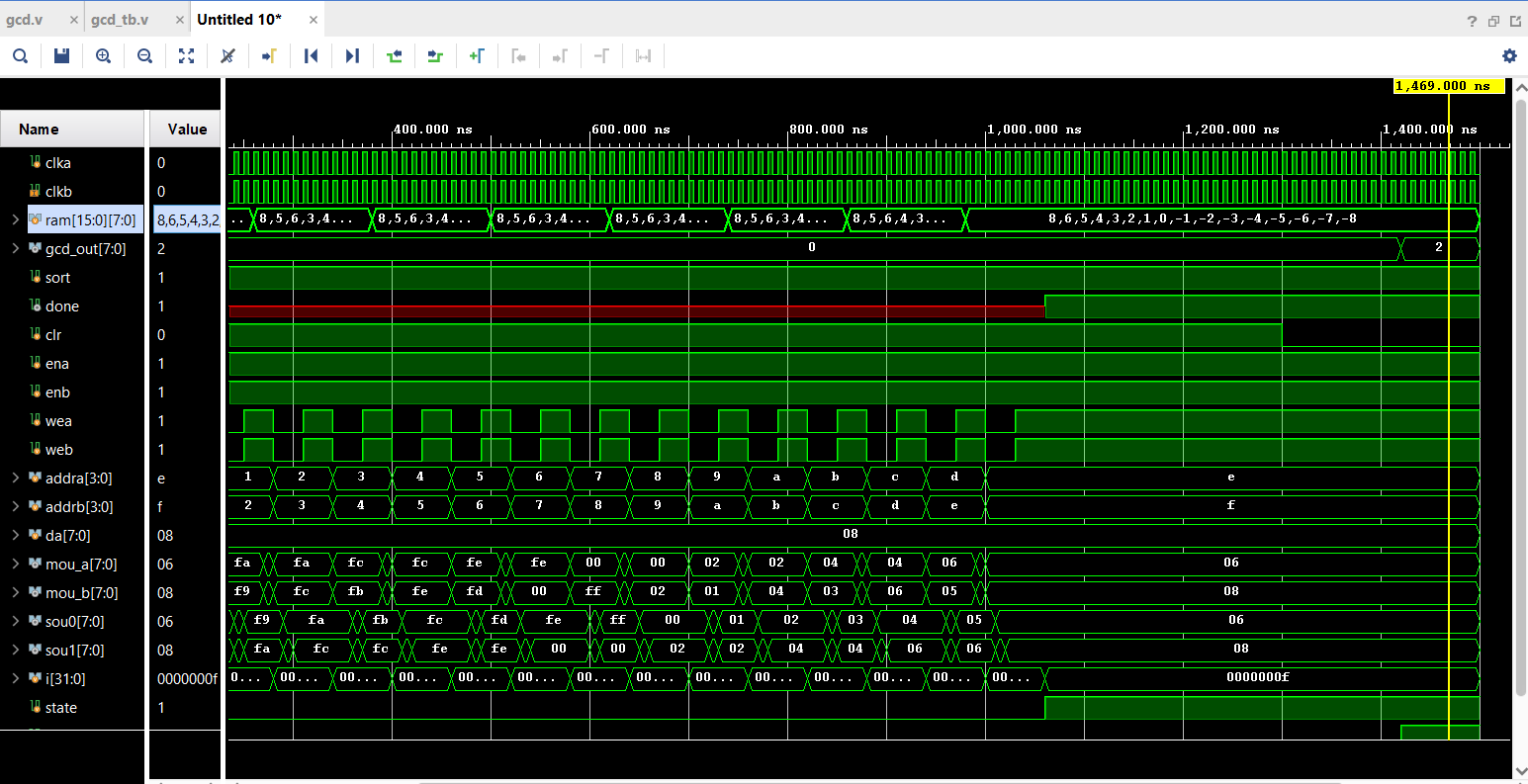
FOR 300 MHZ

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

As can be seen below, the answers for the 2 largest numbers relative to each other vary and there are correct answers. 

metin, elektronik eşyalar, bilgisayar, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

