EE5516 VLSI Architectures for Signal Processing and Machine Learning Lab Report: Experiment 2

Subham Ball
Dept. of System-on-chip Design
Indian Institute Of Technology–Palakkad
152202017@smail.iitpkd.ac.in

***Abstract*—Experiment 2's goal is to implement and validate the design of an architecture that computes the GCD of given two numbers.**

I. INTRODUCTION

To implement an architecture for computing the GCD of two given numbers, we can use the Euclidean algorithm. The steps involved in the algorithm are as follows:

$$GCD(A, B) = GCD(A\%B, B)$$

$$GCD(A, 0) = A$$

1) Set the larger number as the first number and the smaller number as the second number.
2) Divide the first number by the second number and obtain the remainder.
3) If the remainder is zero, the GCD is the second number.
4) If the remainder is non-zero, set the first number as the second number, the second number as the remainder, and go to step 2.

We can implement the architecture using hardware description languages (HDL) such as Verilog or VHDL.

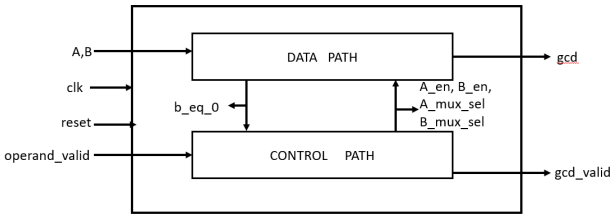

Fig. 1. Block Diagram

II. IMPLEMENTION

The architecture diagram is divided into two parts: the control path, which controls the control flow, and the data path, which regulates the data flow. The data path architecture design is shown in Fig-2, which employs two registers A,B, and mux to pick one of the input to register using state from the control path. If the state value is 00, new input is accepted; if the value is 01, register A (A-B) is updated; if the value is 10, the value is swapped; and in the B register, if the value is 0, new input is accepted; otherwise, the value A is loaded into the B register.

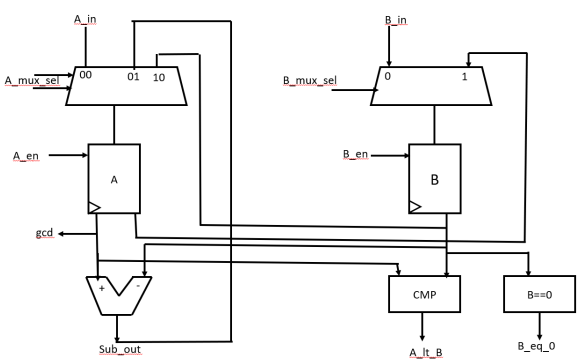

Fig. 2. Data Path

If the B value reaches 0, the comparator used to create the signal B_eq 0 sent as input to the control signal will halt. Figure 3 depicts the architecture's control route, which receives the input operand valid(qualifier), B eq 0, and a signal less signal that states A B and produces the output as A MUX_SEL, B MUX_SEL, A_EN, B_EN,GCD_valid (qualifier)

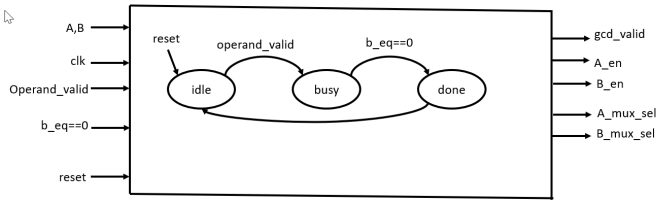

Fig. 3. Control Path

III. Verilog Code

Now let's see the Verilog Code:

design.sv

```
// Code your design here
module gcd_module(
input clk,reset,operand_valid,ack,
input [10:0] A_in ,B_in ,gcd,
output reg gcd_valid);

 // wire declaretion
wire A_en,B_en,B_mux_sel,B_eq_0,A_lt_B;
wire [1:0] A_mux_sel;

// datapath module instantiate
datapath gcd_datapath (.*);
// control path module instantiate
controlpath gcd_controlpath (.*);

endmodule

module datapath(
input clk ,
input [10:0] A_in,B_in ,
input A_en ,B_en,
input [1:0] A_mux_sel ,
input B_mux_sel ,
output reg [10:0] gcd ,
output B_eq_0 ,
output A_lt_B);

// reg and wire declaretion
reg [10:0] A ;
reg [10:0] B ;
wire [10:0] delayed_A ;
wire [10:0] sub_out ;
wire [10:0] A_mux_out ;
wire [10:0] B_mux_out ;

assign B_eq_0 = ( B == 0);
assign A_lt_B = ( A < B );
assign sub_out = A - B ;
assign delayed_A = A ;
assign A_mux_out = A_mux_sel [0]? ( A_mux_sel [1]? A : sub_out ) : ( A_mux_sel [1]? B : A_in );
assign B_mux_out = B_mux_sel ? A : B_in ;

  always@ ( posedge clk )
begin
if ( A_en )
A <= A_mux_out ;
if ( B_en )
B <= B_mux_out ;
gcd <= delayed_A ;
end

endmodule

module controlpath (
input clk,reset,ack,
input operand_valid ,
input B_eq_0 ,
input A_lt_B ,
output reg A_en ,
output reg B_en ,
output reg [1:0] A_mux_sel ,
output reg B_mux_sel ,
output reg gcd_valid);

parameter IDLE = 2'b00;
parameter BUSY = 2'b01;
parameter DONE = 2'b10;
reg [1:0] state ;
reg [1:0] next_state ;

always@ ( posedge clk )
begin
if ( reset )
state <= 00;
else
state <= next_state ;
end

always@ (*)
begin
case ( state )
IDLE : if ( operand_valid ==1) next_state = BUSY ;
BUSY : if ( B_eq_0 ==1) next_state = DONE ;
DONE : if ( ack ==1) next_state = IDLE ;
endcase
end


always@ (*)
begin
case ( state )
IDLE :
begin
A_en = 1'b1;
B_en = 1'b1;
A_mux_sel = 2'b00 ;
B_mux_sel = 1'b0 ;
end
BUSY :
begin
if ( A_lt_B )
begin
A_en = 1'b1;
B_en = 1'b1;
A_mux_sel = 2'b10;
B_mux_sel = 1'b1;
end
else if (~ B_eq_0 )
begin
A_mux_sel = 2'b01;
A_en = 1'b1;
B_en = 1'b0;
end
if ( B_eq_0 )
begin
A_mux_sel = 2'b11;
end
end
endcase
end

assign gcd_valid = ( state == DONE );

endmodule
```

Fig. 4. RTL code

Now let's see the testbench for the same:

testbench.sv

```
// Code your testbench here
// or browse Examples
module gcd_test ;
reg [10:0] A_in , B_in , gcd ;
reg clk , reset , operand_valid , ack , gcd_valid ;
gcd_module DUT_gcd (.*);
always #3 clk = ~ clk ;
initial
begin
$dumpvars (0);
$dumpfile (" dump .vcd");
clk = 1'b0;
ack = 1'b0;
reset = 0;
#3;
reset = 1;
#3;
reset = 0;
#3;
operand_valid = 1;
A_in = 48;
B_in = 32;
#40;
operand_valid = 0;
ack = 1'b1 ;
#200;
$finish ;
end
endmodule
```

Fig. 5. Testbench OF DUT

IV. Experimental results

Now, let's see the output for the simulation.

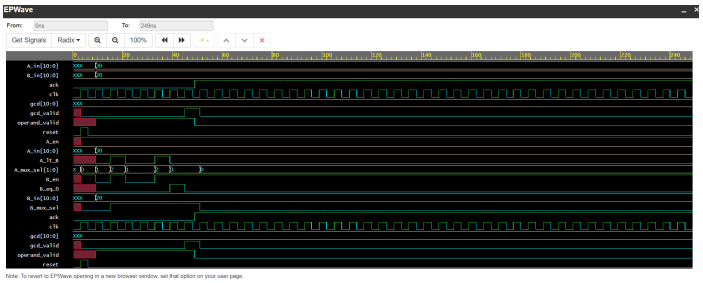

Fig. 6. Output Waveform

V. Conclusion

In conclusion, GCD computing is an essential part of VLSI design since it helps to reduce the number of clock cycles required for specific tasks. The GCD may be computed using a variety of techniques, including the Euclidean algorithm and the binary GCD algorithm. The algorithm chosen is determined by the unique needs of the VLSI design, such as speed and accuracy.We have successfully verified the working of the Euclidean algorithm in this experiment.The link to the EDA Playground is: playground link