**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #7**

**Due day: 4/21/2023**

**Instruction:**

1. **Push the answer sheet to GitHub in word file.**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. (Bonus) Write two Verilog modules to design a 4-bits multiplier which implements Booth’s algorithm and one of multiplication algorithms from 3 versions shown in the handout of *Lec06-alu.pdf,* respectively.

**Answer:**

**Github:** <https://github.com/KhandokerSamiulHoque/488bonus.git> **1.1**

**Design:**

module boothAlgo\_mul(input [3:0] multiplicand,input [3:0] multiplier,output reg [7:0] product);

reg [4:0] s;

reg [4:0] a;

reg [2:0] i;

always @(\*) begin

s[4:0] = {1'b0, multiplier};

a[4:0] = {1'b0, multiplicand, 2'b0};

product = 0;

for (i = 0; i < 4; i = i + 1) begin

if (s[1:0] == 2'b01)

begin

product = product + a[4:0];

end

else if (s[1:0] == 2'b10) begin

product = product - a[4:0];

end

if (product[0] == 1) begin

s[1:0] = s[1:0] + 2'b10;

end

s[4:1] = s[3:0];

s[0] = a[0];

a[4:1] = a[3:0];

a[0] = product[0];

product = product >> 1;

end

end

endmodule

**Testbench:**

module boothAlgo\_mul\_tb;

reg [3:0] multiplicand;

reg [3:0] multiplier;

wire [7:0] product;

boothAlgo\_mul ee488prob1 (.multiplicand(multiplicand),.multiplier(multiplier),.product(product));

reg clock = 0;

always

#5

clock = ~clock;

initial begin

$dumpfile("boothAlgo\_mul\_tb.vcd");

$dumpvars(0, boothAlgo\_mul\_tb);

multiplicand = 4'b0010;

multiplier = 4'b0011;

#10;

multiplicand = 4'b0010;

multiplier = 4'b0110;

#10;

multiplicand = 4'b0010;

multiplier = 4'b1100;

#10;

multiplicand = 4'b0010;

multiplier = 4'b1000;

#10;

multiplicand = 4'b0010;

multiplier = 4'b0000;

#10;

$finish;

end

always @(posedge clock) begin

$display("multiplicand = %b, multiplier = %b, product = %b", multiplicand, multiplier, product);

end

endmodule **1.2**

**Design:**

module multiplier\_algorithm(input [31:0] multiplicand,input [31:0] multiplier,output reg [63:0] product);

integer i;

always @(\*) begin

product = 0;

for (i = 0; i < 32; i = i + 1) begin

if (product[0]) begin

product[63:32] = product[63:32] + multiplicand;

end

product = {product[62:0], product[63]};

if (multiplier[31 - i]) begin

product[31:0] = product[31:0] + multiplicand;

end

end

end

endmodule

**Testbench:**

module tb\_multiplier\_algorithm;

reg [31:0] multiplicand;

reg [31:0] multiplier;

wire [63:0] product;

multiplier\_algorithm uut(.multiplicand(multiplicand),.multiplier(multiplier),

.product(product));

initial begin

$dumpfile("tb\_multiplier\_algorithm.vcd");

$dumpvars;

$monitor("Time = %t | multiplicand = %h | multiplier = %h | product = %h", $time, multiplicand, multiplier, product);

multiplicand = 32'h00000001;

multiplier = 32'h00000011;

#10;

multiplicand = 32'h00000011;

multiplier = 32'h00000101;

#10;

multiplicand = 32'h00000000;

multiplier = 32'h00001100;

#10;

$finish;

end

endmodule

1. (Bonus) Write two Verilog modules to design a 4-bits divisor which implements any two of division algorithms from 3 versions shown in the handout of *Lec07-division.pdf,* respectively.

**Answer:  
  
2.1**

**Design:**

module divisor\_met1 (input clk,input [3:0] dividend,input [3:0] divisor,input start,output [3:0] quotient,output [3:0] remainder,output out);

reg [3:0] dividend\_reg, divisor\_reg, quotient\_reg;

reg [2:0] counter;

reg [4:0] adder\_out;

always @(posedge clk) begin

if (start) begin

dividend\_reg <= {dividend, 4'b0};

divisor\_reg <= {divisor, 4'b0};

quotient\_reg <= 4'b0;

counter <= 3'b0;

adder\_out <= 4'b0;

end

else if (counter < 4)

begin

adder\_out <= dividend\_reg + (~divisor\_reg + 1);

if (adder\_out[3] == 1)

begin

quotient\_reg[counter] <= 0;

adder\_out <= adder\_out + divisor\_reg;

end else

begin

quotient\_reg[counter] <= 1;

end

dividend\_reg <= adder\_out[3:0];

counter <= counter + 1;

end

end

assign quotient = quotient\_reg;

assign remainder = dividend\_reg[3:0];

assign out = (counter == 4);

endmodule

**Testbench:**module divisor\_met1\_tb;

reg clk;

reg [3:0] dividend;

reg [3:0] divisor;

reg start;

wire [3:0] quotient;

wire [3:0] remainder;

wire out;

divisor\_met1 ee488prob2.1 (.clk(clk),.dividend(dividend),.divisor(divisor),.start(start),.quotient(quotient),.remainder(remainder),

.out(out)

);

initial begin

$dumpfile("divisor\_met1\_tb.vcd");

$dumpvars(0,divisor\_met1\_tb);

clk = 0;

dividend = 4'b1010;

divisor = 4'b1101;

start = 0;

#10

start = 1;

#10

start = 0;

#20

dividend = 4'b0000;

#20

divisor = 4'b0000;

#100

$finish;

end

always

#5

clk = ~clk;

endmodule

**2.2**

**Design:**

module divisor\_meth(input [3:0] dividend,input [3:0] divisor,output reg [3:0] quotient);

reg [3:0] remainder;

integer i;

always @(\*) begin

quotient = 4'b0000;

remainder = dividend;

for (i = 3; i >= 0; i = i-1) begin

remainder = remainder << 1;

quotient[i] = remainder >= divisor;

if (quotient[i]) remainder = remainder - divisor;

end

end

endmodule

**Testbench:**

module divisor\_meth\_tb;

reg [3:0] dividend;

reg [3:0] divisor;

wire [3:0] quotient;

divisor\_meth uut (

.dividend(dividend),

.divisor(divisor),

.quotient(quotient)

);

initial begin

$dumpfile("divisor\_meth\_tb.vcd");

$dumpvars(0, divisor\_meth\_tb);

dividend = 4'b1001;

divisor = 4'b0011;

#10;

$display("Quotient: %b", quotient);

dividend = 4'b1101;

divisor = 4'b0101;

#10;

$display("Quotient: %b", quotient);

dividend = 4'b1111;

divisor = 4'b0111;

#10;

$display("Quotient: %b", quotient);

dividend = 4'b1011;

divisor = 4'b1001;

#10;

$display("Quotient: %b", quotient);

end

endmodule