**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.

**4-bits multiplier:**

module booth\_multiplier(

input signed [3:0] multiplicand,

input signed [3:0] multiplier,

output reg signed [7:0] product

);

reg signed [7:0] product\_reg;

reg [4:0] m;

reg [4:0] a;

reg [2:0] q;

always @(\*) begin

product\_reg = 8'b0;

m = {1'b0, multiplicand};

a = {4'b0, multiplier};

q = 3'b0;

repeat(4) begin

// check last two bits of q and E1

case ({q[1], q[0], q[2]})

3'b001, 3'b010 : product\_reg = product\_reg - m;

3'b100, 3'b011 : product\_reg = product\_reg + m;

default : begin end

endcase

// shift right

q = {a[0], q[2:1]};

a = {a[4], a[3:1]};

end

// check if multiplier is negative

if (multiplier[3] == 1) begin

product\_reg = -product\_reg;

end

product = product\_reg; // assign the result to the output

end

endmodule

**Testbench:**

module booth\_multiplier\_tb;

// Inputs

reg [3:0] multiplicand;

reg [3:0] multiplier;

// Outputs

wire [7:0] product;

// Instantiate the Unit Under Test (UUT)

booth\_multiplier uut (

.multiplicand(multiplicand),

.multiplier(multiplier),

.product(product)

);

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus

initial begin

// Create a VCD dump file

$dumpfile("booth\_multiplier.vcd");

$dumpvars(0, booth\_multiplier\_tb);

// test 1

multiplicand = 4'b0101;

multiplier = 4'b0011;

#10;

$display("Product = %d", product);

// test 2

multiplicand = 4'b1001;

multiplier = 4'b0010;

#10;

$display("Product = %d", product);

// test 3

multiplicand = 4'b0110;

multiplier = 4'b1010;

#10;

$display("Product = %d", product);

// test 4

multiplicand = 4'b0010;

multiplier = 4'b1001;

#10;

$display("Product = %d", product);

// test 5

multiplicand = 4'b1111;

multiplier = 4'b0101;

#10;

$display("Product = %d", product);

// test 6

multiplicand = 4'b0000;

multiplier = 4'b0001;

#10;

$display("Product = %d", product);

// test 7

multiplicand = 4'b1111;

multiplier = 4'b1111;

#10;

$display("Product = %d", product);

// test 8

multiplicand = 4'b1010;

multiplier = 4'b0101;

#10;

$display("Product = %d", product);

// test 9

multiplicand = 4'b1101;

multiplier = 4'b1011;

#10;

$display("Product = %d", product);

// test 10

multiplicand = 4'b0010;

multiplier = 4'b0001;

#10;

$display("Product = %d", product);

// End simulation

#10 $finish;

end

// Clock driver

always #2.5 clk = ~clk;

endmodule

**Multiplication algorithms from 3 versions:**

***Design:***

module booth\_multiplier(clk, rst, multiplicand, multiplier, product);

input clk, rst;

input [3:0] multiplicand, multiplier;

output reg [7:0] product;

reg [4:0] counter;

reg [3:0] complement;

initial begin

counter = 4;

complement = ~multiplicand + 1;

product = 0;

end

always @(posedge clk) begin

if (rst) begin

counter <= 4;

complement <= ~multiplicand + 1;

product <= 0;

end

else begin

case ({multiplier[3], multiplier[2], multiplier[1]}) // examine three bits at a time

3'b000, 3'b111: product <= {product[6:0], product[7]}; // shift right

3'b001, 3'b100: product <= product; // no change

3'b010: product <= product + complement; // add complement

3'b101: product <= product - complement; // subtract complement

endcase

counter <= counter - 1;

end

end

endmodule

***Testbench :***

*module booth\_multiplier\_tb();*

*reg clk, rst;*

*reg [3:0] multiplicand, multiplier;*

*wire [7:0] product;*

*booth\_multiplier dut(.clk(clk),*

*.rst(rst),*

*.multiplicand(multiplicand),*

*.multiplier(multiplier),*

*.product(product));*

*initial begin*

*clk = 0;*

*rst = 0;*

*multiplicand = 4'b0101;*

*multiplier = 4'b1011;*

*#10 rst = 1;*

*#10 rst = 0;*

*#10 multiplicand = 4'b1111;*

*#10 multiplier = 4'b0001;*

*#10 $finish;*

*end*

*always #5 clk = ~clk;*

*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.

**Design 1:**

**Dividend: 101101**

**Divisor: 0101**

module booth\_division\_v3 (

input [5:0] dividend,

input [3:0] divisor,

output reg [1:0] quotient,

output reg [3:0] remainder

);

reg [3:0] complement;

reg [2:0] state = 0;

reg [5:0] dividend\_shifted;

always @(\*) begin

// Initialize complement register

complement = ~divisor + 1;

// Append 0s to dividend

dividend\_shifted = {2'b00, dividend};

// Perform division using Booth's algorithm version 3

for (int i = 0; i < 6; i = i + 3) begin

case (dividend\_shifted[i+2:i])

3'b000, 3'b111: begin

quotient = quotient << 1;

dividend\_shifted = dividend\_shifted << 1;

end

3'b001, 3'b110: begin

quotient = quotient;

dividend\_shifted = dividend\_shifted << 1;

end

3'b010: begin

quotient = quotient << 1;

dividend\_shifted = dividend\_shifted + complement;

dividend\_shifted = dividend\_shifted << 1;

end

3'b101: begin

quotient = quotient << 1;

quotient[0] = 1;

dividend\_shifted = dividend\_shifted - complement;

dividend\_shifted = dividend\_shifted << 1;

end

endcase

end

// Remove appended 0s from dividend to obtain remainder

remainder = dividend\_shifted[5:2];

end

endmodule

**Testbench:**

module booth\_division\_v3\_tb;

reg [5:0] dividend;

reg [3:0] divisor;

wire [1:0] quotient;

wire [3:0] remainder;

booth\_division\_v3 dut (

.dividend(dividend),

.divisor(divisor),

.quotient(quotient),

.remainder(remainder)

);

initial begin

dividend = 6'b101101;

divisor = 4'b0101;

#10;

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

$display("Remainder: %b", remainder);

$finish;

end

endmodule

**Design2**

**Dividend: 1101011**

**Divisor: 1001**

module booth\_division\_v3 (

input [7:0] dividend,

input [3:0] divisor,

output reg [3:0] quotient,

output reg [3:0] remainder

);

reg [3:0] complement;

reg [2:0] state = 0;

reg [7:0] dividend\_shifted;

always @(\*) begin

// Initialize complement register

complement = ~divisor + 1;

// Append 0s to dividend

dividend\_shifted = {2'b00, dividend};

// Perform division using Booth's algorithm version 3

for (int i = 0; i < 8; i = i + 3) begin

case (dividend\_shifted[i+2:i])

3'b000, 3'b111: begin

quotient = quotient << 1;

dividend\_shifted = dividend\_shifted << 1;

end

3'b001, 3'b110: begin

quotient = quotient;

dividend\_shifted = dividend\_shifted << 1;

end

3'b010: begin

quotient = quotient << 1;

dividend\_shifted = dividend\_shifted + complement;

dividend\_shifted = dividend\_shifted << 1;

end

3'b101: begin

quotient = quotient << 1;

quotient[0] = 1;

dividend\_shifted = dividend\_shifted - complement;

dividend\_shifted = dividend\_shifted << 1;

end

endcase

end

// Remove appended 0s from dividend to obtain remainder

remainder = dividend\_shifted[7:4];

end

endmodule

**Testbench**

module booth\_division\_v3\_tb;

reg [7:0] dividend;

reg [3:0] divisor;

wire [3:0] quotient;

wire [3:0] remainder;

booth\_division\_v3 dut (

.dividend(dividend),

.divisor(divisor),

.quotient(quotient),

.remainder(remainder)

);

initial begin

dividend = 8'b11010110;

divisor = 4'b1001;

#10;

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

$display("Remainder: %b", remainder);

$finish;

end

endmodule