**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.

Design module: Booth Multiplier:

module Booth\_multiplier(input signed [3:0] multiplicand, input signed [3:0] multiplier, output signed [7:0] product);

reg signed [3:0] m\_reg;

always @(\*) m\_reg = {1'b0, multiplier};

reg signed [7:0] p\_reg;

always @(\*) p\_reg = {4'b0, multiplicand};

reg control = 1'b0;

integer i;

always @(\*) begin

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

if (m\_reg[1:0] == 2'b01) begin

if (control == 1'b0) p\_reg = p\_reg - {4'b0, multiplicand};

else p\_reg = p\_reg + {4'b0, multiplicand};

end

else if (m\_reg[1:0] == 2'b10) begin

if (control == 1'b0) p\_reg = p\_reg + {4'b0, multiplicand};

else p\_reg = p\_reg - {4'b0, multiplicand};

end

if (m\_reg[0] == 1'b0) begin

m\_reg = {m\_reg[2:0], m\_reg[3]};

control = m\_reg[3];

end

else begin

m\_reg = {m\_reg[2:0], m\_reg[3]};

control = 1'b0;

end

p\_reg = {p\_reg[6:0], p\_reg[7]};

end

end

assign product = p\_reg;

endmodule

**Testbench:**

module testbench;

reg signed [3:0] multiplicand = 4'sb1010;

reg signed [3:0] multiplier = 4'sb1111;

wire signed [7:0] product;

Booth\_multiplier dut(

.multiplicand(multiplicand),

.multiplier(multiplier),

.product(product)

);

reg clk = 1'b0;

always #10 clk = ~clk;

initial begin

$dumpfile("tb.vcd");

$dumpvars;

$monitor("Time=%0t multiplicand=%b multiplier=%b product=%b", $time, multiplicand, multiplier, product);

#10 multiplicand = 4'sb0001; multiplier = 4'sb0010;

#10 multiplicand = 4'sb0100; multiplier = 4'sb0011;

#10 multiplicand = 4'sb0011; multiplier = 4'sb0110;

#10 multiplicand = 4'sb1111; multiplier = 4'sb1101;

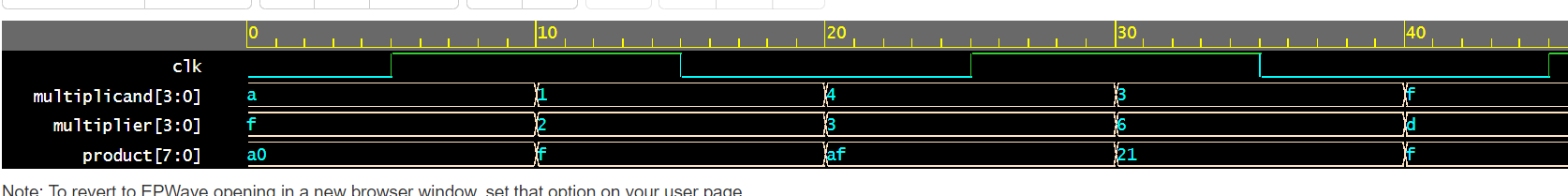
#10 $finish;

end

always #5 clk = ~clk;

endmodule

waveform:



**Multiplier algorithm ver-3:**

Design code:

module booth\_mulvar\_3(clk, reset, multiplicand, multiplier, product);

input clk, reset;

input signed [3:0] multiplicand, multiplier;

output signed [7:0] product;

reg signed [7:0] acc = 8'b0000\_0000;

reg [1:0] state;

wire [4:0] ext\_multiplier;

assign ext\_multiplier = {multiplier[3], multiplier};

always @(posedge clk or posedge reset) begin

if (reset) begin

acc <= 8'b0000\_0000;

state <= 2'b00;

end

else begin

case (ext\_multiplier[1:0])

2'b01:

begin

acc <= acc - multiplicand;

state <= 2'b01;

end

2'b10:

begin

acc <= acc + multiplicand;

state <= 2'b01;

end

2'b11:

begin

acc <= acc + (multiplicand << 1);

state <= 2'b10;

end

2'b00:

begin

state <= 2'b10;

end

default:

begin

acc <= acc - (multiplicand << 1);

state <= 2'b10;

end

endcase

end

end

assign product = acc;

endmodule

Testbench:

module tb;

reg clk, reset;

reg signed [3:0] multiplicand, multiplier;

wire signed [7:0] product;

booth\_mulvar\_3 U(

.clk(clk),

.reset(reset),

.multiplicand(multiplicand),

.multiplier(multiplier),

.product(product)

);

initial begin

$dumpfile("tb.vcd");

$dumpvars;

clk = 0;

reset = 0;

multiplicand = 4'b1010;

multiplier = 4'b1111;

#10 reset = 0;

#10 multiplicand = 4'b0001;

#10 multiplier = 4'b0010;

#10 multiplicand = 4'b0100;

#10 multiplier = 4'b0011;

#10 multiplicand = 4'b0011;

#10 multiplier = 4'b0110;

#10 multiplicand = 4'b1111;

#10 multiplier = 4'b1101;

#10 $finish;

end

always #5 clk = ~clk;

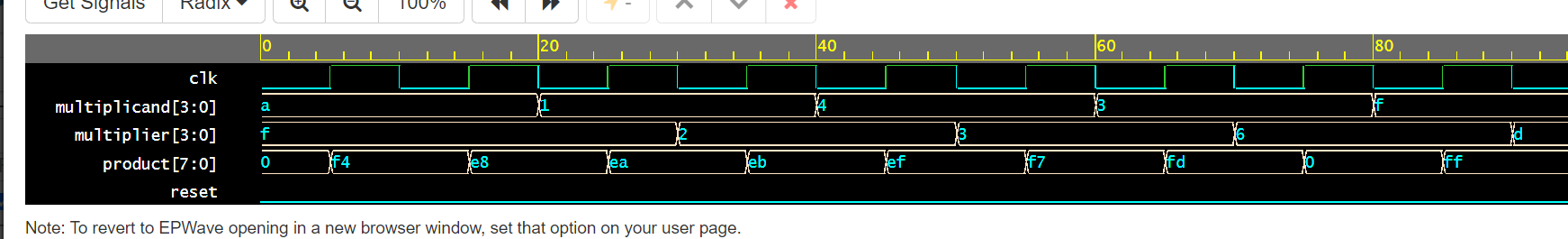
always @(posedge clk) begin

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

end

endmodule

Waveform:



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 code for restoring division Algorithm**:

module divider\_restor(input wire [31:0] dividend,

input wire [31:0] divisor,

input clk,

output reg [31:0] quotient,

output reg [31:0] remainder,

output reg done,

input rst);

reg [31:0] dividend\_reg;

reg [31:0] shifted\_divisor;

reg [31:0] partial\_quotient;

integer i;

always @(posedge clk) begin

if (done) begin

quotient <= 0;

remainder <= 0;

done <= 0;

dividend\_reg <= 0;

shifted\_divisor <= 0;

partial\_quotient <= 0;

end else begin

if (i == 32) begin

quotient <= partial\_quotient;

remainder <= dividend\_reg;

done <= 1;

end else begin

shifted\_divisor <= divisor << i;

if (dividend\_reg >= shifted\_divisor) begin

partial\_quotient[i] <= 1;

dividend\_reg <= dividend\_reg - shifted\_divisor;

end else begin

partial\_quotient[i] <= 0;

end

i <= i + 1;

end

end

end

initial begin

quotient <= 0;

remainder <= 0;

done <= 0;

dividend\_reg <= dividend;

shifted\_divisor <= 0;

partial\_quotient <= 0;

i <= 0;

end

endmodule

**Testbench:**

module tb;

reg [31:0] dividend;

reg [31:0] divisor;

reg clk;

reg rst;

wire [31:0] quotient;

wire [31:0] remainder;

wire done;

divider\_restor u(

.dividend(dividend),

.divisor(divisor),

.clk(clk),

.rst(rst),

.quotient(quotient),

.remainder(remainder),

.done(done)

);

always #5 clk = ~clk;

always @(posedge clk) begin

$display("dividend=%d, divisor=%d, quotient=%d, remainder=%d, done=%d", dividend, divisor, quotient, remainder, done);

end

initial begin

$dumpfile("tb.vcd");

$dumpvars;

rst = 1;

#5 rst = 0;

// Test case 1

dividend = 100;

divisor = 3;

#10;

// Test case 2

dividend = 500;

divisor = 7;

#10;

// Test case 3

dividend = 123456;

divisor = 654;

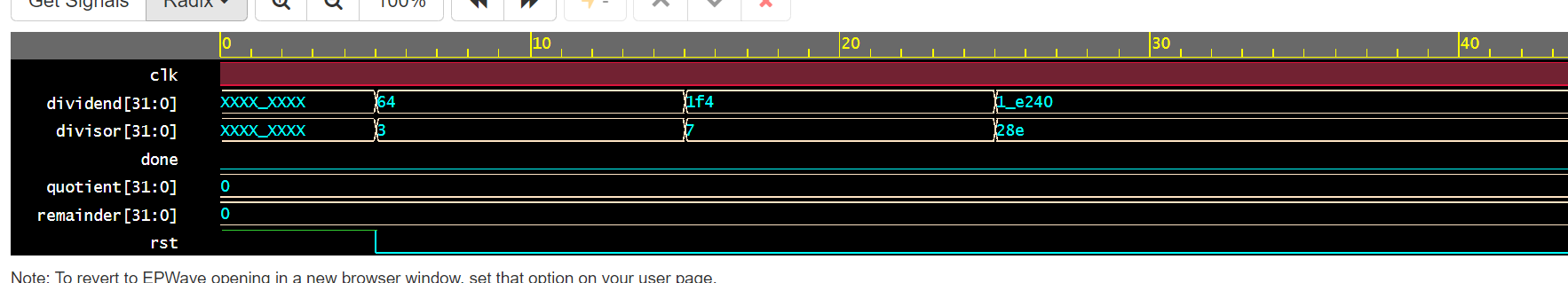
#10;

#10 $finish();

end

endmodule

waveform



Design module for non-restoring div:

module division\_non\_restor (

input [31:0] dividend,

input [31:0] divisor,

input clk,

output reg [31:0] quotient,

output reg [31:0] remainder,

output reg done,

input rst);

reg [31:0] partial\_quotient;

reg [63:0] temp;

reg [31:0] count;

reg [31:0] i;

reg [31:0] j;

reg [31:0] k;

always @(posedge clk) begin

if (done == 1'b0) begin

if (j == 0) begin

if (temp >= divisor) begin

partial\_quotient <= 32'h1;

end else begin

partial\_quotient <= 32'h0;

end

j <= 32'h1;

end else begin

j <= j + 32'h1;

if (j == 32'h20) begin

j <= 32'h0;

k <= 32'h1;

end else begin

k <= k << 1;

end

if (temp >= divisor\*k) begin

partial\_quotient <= partial\_quotient + k;

temp <= temp - divisor\*k;

end

end

if (i == 32'h1f) begin

i <= 32'h0;

quotient <= partial\_quotient;

remainder <= temp[31:0];

done <= 1'b1;

end else begin

i <= i + 32'h1;

temp <= {temp[30:0], dividend[i]};

end

end

end

endmodule

Testbench:

module tb;

reg [31:0] dividend;

reg [31:0] divisor;

reg clk;

reg rst;

wire [31:0] quotient;

wire [31:0] remainder;

wire done;

division\_non\_restor U(

.dividend(dividend[31:0]),

.divisor(divisor[31:0]),

.clk(clk),

.rst(rst),

.quotient(quotient),

.remainder(remainder),

.done(done)

);

always #5 clk = ~clk;

initial begin

$dumpfile("tb.vcd");

$dumpvars;

rst = 1;

#10 rst = 0;

// Test case 1

dividend = 32'h64;

divisor = 32'h3;

#100;

// Test case 2

dividend = 32'h1f4;

divisor = 32'h7;

#100;

// Test case 3

dividend = 32'h1e240;

divisor = 32'h28e;

#100;

// End simulation

#100 $finish;

end

endmodule

waveform:

