Lab Report - Experiment 1

EE21B015 - Ananya Das
EE21B098 - Sailendra
Batch 44.

CONTENTS

4. Result

Aim

To study the 4-bit serial-parallel multiplier and Booths algorithm for multiplication.

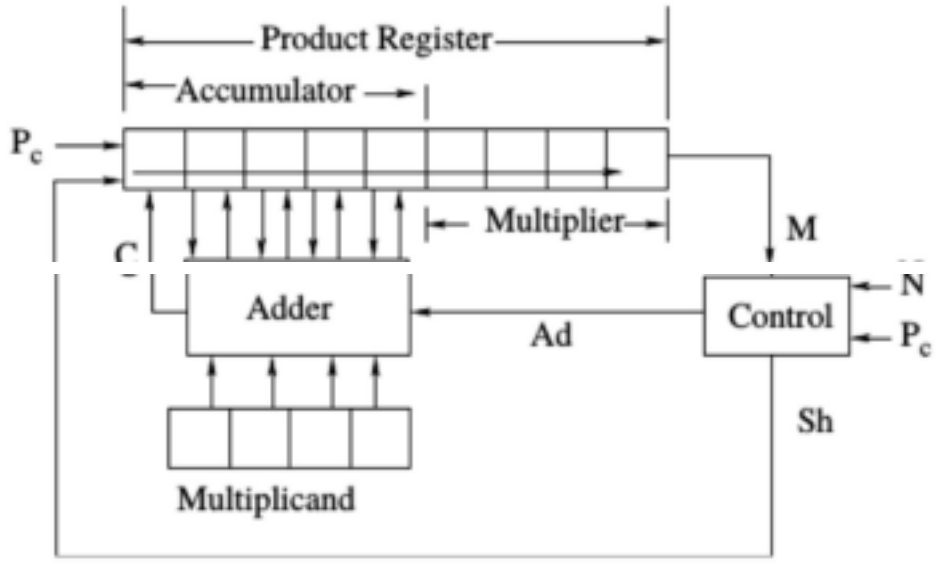
1) Serial Parallel Multiplier

In this experiment, for academic reasons, we implement and then compare the serial-parallel multiplier with the Booth's algorithm. multiply the multiplicand by Least Significant Digit of multiplier, shift right, then repeat for the next digit to write-out the product beneath the first product, keep doing till Most Significant Digit and add them now. The only difference is that for each multiplication above, use repeated addition and use the same accumulator to hold the product.

Multipliers of unsigned numbers generally fall in one of three categories: array (parallel), serial and serial - parallel. One of the two operands in a serial- parallel multiplier is loaded in parallel while the other operand is fed serially. Serial - parallel multipliers are used for hardware simplicity and moderate speed. Here, an unsigned 4-bit multiplier has to be implemented on an FPGA board. The numbers can be hard coded in your program. The results should appear on the LEDs on the FPGA board.

When an add signal (Ad) is given, the adder outputs are transferred to the accumulator by the next clock pulse (Pc) and this corresponds to adding the multiplicand to the accumulator. An extra bit at the left end of the product register temporarily stores any carry generated when the multiplicand is added to the accumulator (Convince yourself that this extra location is required for this multiplier implementation). Sh corresponds to a shift signal while N is used to start the operation (in particular, N is set to 1). The current multiplier bit is denoted by M while C denotes carry.

ALGORITHM



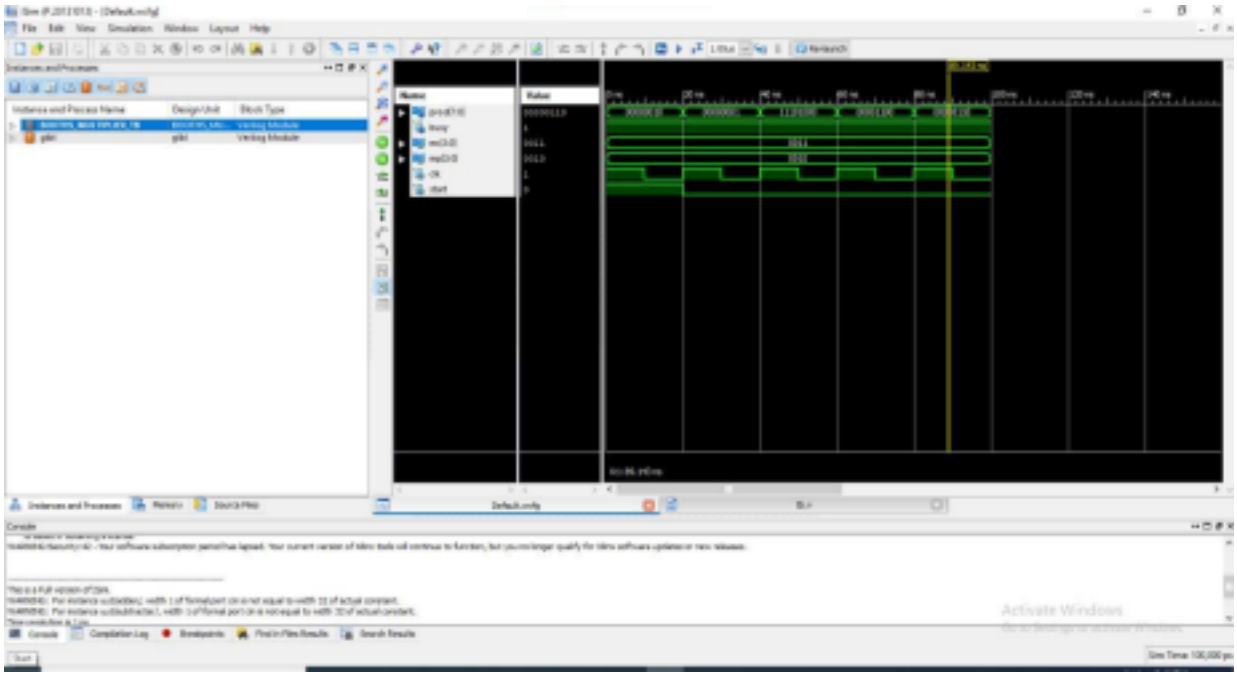
CODE

```
module noradd(
 input [3:0]A,
 input [3:0]B,
 output reg [7:0]product   );
always@(A or B)
begin

 product = 8'b00000000;   if(B[0] ==
1'b1)
 product = product + (A<<0);   if(B[1] == 1'b1)
 product = product + (A<<1);   if(B[2] == 1'b1)
 product = product + (A<<2);   if(B[3] == 1'b1)
 product = product + (A<<3);

end
endmodule
```

TEST BENCH

```
module testi;

 // Inputs
 reg [3:0] A;
 reg [3:0] B;

 // Outputs
 wire [7:0] product;




 // Instantiate the Unit Under Test (UUT)   noradd uut (
 .A(A),
 .B(B),
 .product(product)
 );

 initial begin
 // Initialize Inputs
 A = 0;
 B = 0;

 // Wait 100 ns for global reset to finish   #100;

 // Add stimulus here
 A = 1;
 B = 1;
 end

endmodule
```

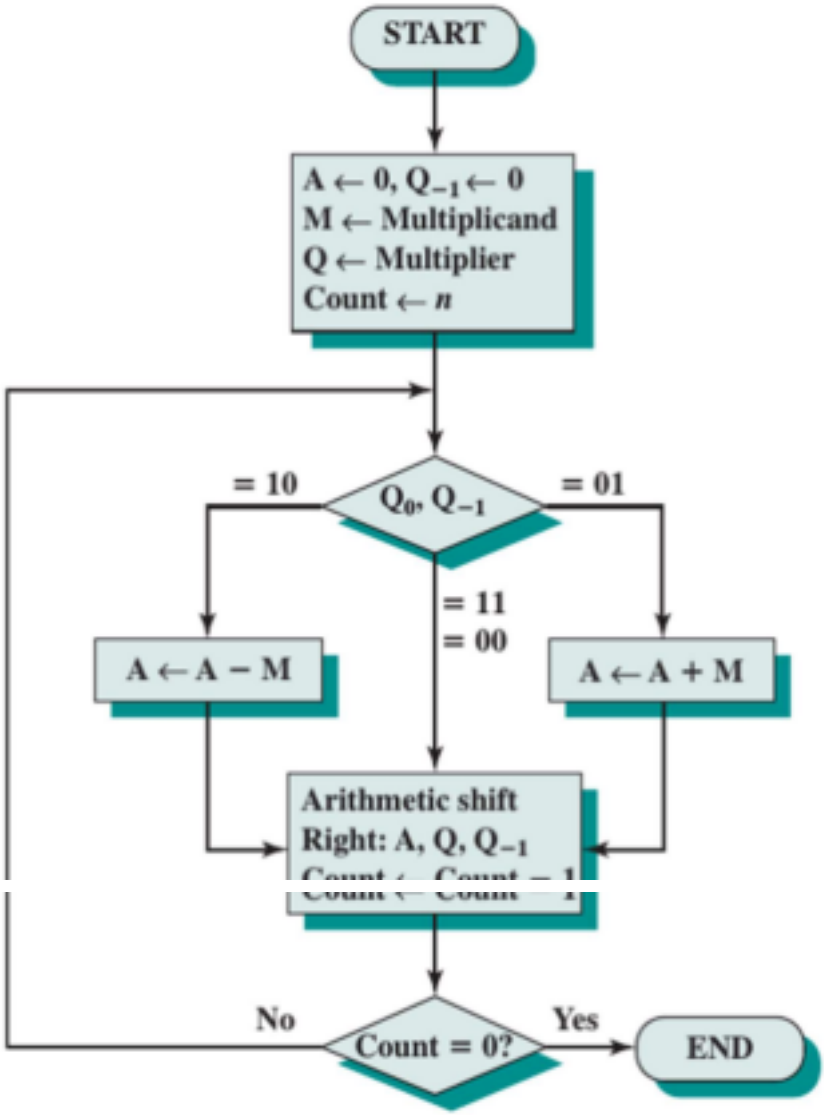
OUTPUT



2) Booth’s Multiplication Algorithm

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. Booth's algorithm performs fewer additions and subtractions than the normal multiplication algorithm.

ALGORITHM



CODE

```
module BOOTHS_MULTIPLIER(
 output [7:0] prod,
 output busy,
 input [3:0] mc,
 input [3:0] mp,
 input clk,
 input start
```

```
);
reg [3:0] A, Q, M; // all registers are of 4 bits   reg Q_1;
reg [2:0] count;
wire [3:0] sum, difference;

always @(posedge clk)
begin
if (start)
begin
A <= 1'b0;
M <= mc;
Q <= mp;
Q_1 <= 1'b0; // bit written to the left of lsb of  number to be multiplied
count <= 3'b0;
end
else
begin
case ({Q[0], Q_1})
2'b0_1 : {A, Q, Q_1} <= {sum[3], sum, Q};
2'b1_0 : {A, Q, Q_1} <= {difference[3],  difference, Q};
default: {A, Q, Q_1} <= {A[3], A, Q};   endcase
count <= count + 1'b1;
          end


                    end
end
alu adder(sum, A, M, 0); // adder
alu subtracter(difference, A,~M, 1); //subtractor using 2's  compliment
assign prod = {A, Q}; // make it fill up the arguments   assign busy = (count < 5);
endmodule

// The following is an alu.It is an adder, but capable of subtraction:
// Recall that subtraction means adding the two's complement-- a - b =  a + (-b) = a + (inverted b
+ 1)
// The 1 will be coming in as cin (carry-in)
```

```
module alu(out, a, b, cin);
 output [3:0] out;
 input [3:0] a;
 input [3:0] b;
 input cin;
 assign out = a + b + cin;
endmodule
```

TEST BENCH

```
module BOOTHS_MULTIPLIER_TB;

 // Inputs
 reg [3:0] mc;
 reg [3:0] mp;
 reg clk;
 reg start;

 // Outputs
 wire [7:0] prod;
 wire busy;
```



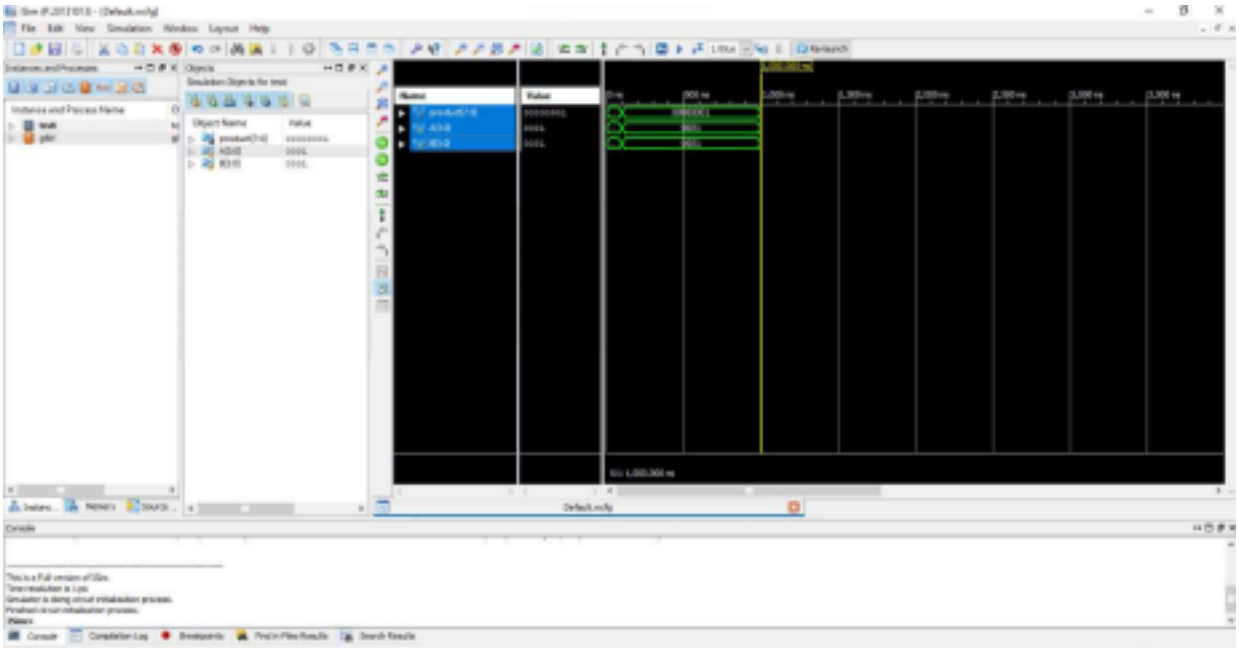
```
        // Instantiate the Unit Under Test (UUT)
BOOTHS_MULTIPLIER uut (
 .prod(prod),
 .busy(busy),
 .mc(mc),
 .mp(mp),
 .clk(clk),
 .start(start)
 );
 initial begin
 // Initialize Inputs
```

```
mc = 4'b0011;
mp = 4'b0010;
clk = 1;
start = 1;
#10 clk = ~clk;
#10 clk = ~clk;
start = 0;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
#10 clk = ~clk;
$finish;
end
initial begin

$dumpfile("BOOTHS.vcd");
$dumpvars(0,BOOTHS_MULTIPLIER_TB);
```

OUTPUT

```
end
endmodule
```

OUTPUT



RESULT

- Both serial parallel multiplier and Booth’s algorithm for multiplication were implemented and hence compared.
- The Verilog code was implemented and the test bench was simulated giving the desired output result