EE2016 Micropprocessor Lab & Theory Aug-Nov. 2019

EE Department, IIT, Madras.

Experiment 9: Implementation and Performance Comparison of 4-bit Serial-Parallel Multiplier with Booth's Algorithm in FPGA (Xilinx's Spartan 3E Board)

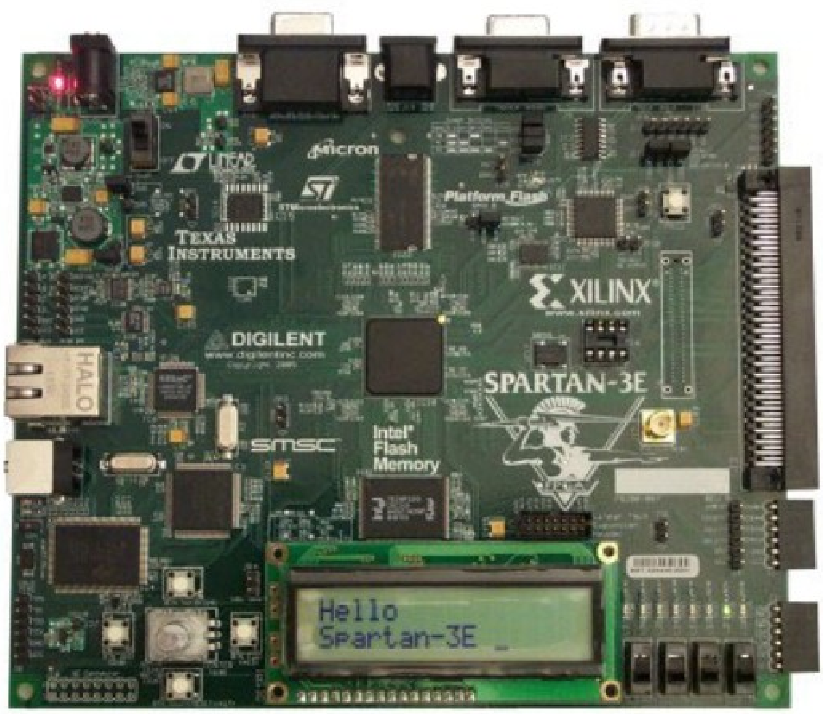
1 Aim

1. To study the 4-bit serial-parallel multiplier and Booths algorithm for multiplication
2. To implement both the above in FPGA platform
3. To demonstrate its working given a test set, by writing a test bench code to display the output in LEDs
4. To compare the performance of both the algorithms in terms of number of clock cycles, given the same set of multiplicand and multiplier

2 Equipments, Hardwares / Softwares Required

The list of equipments, components required are:

1. Spartan board



2. Xilinx Software (IDE)
3. PC hosting the Xilinx Software and JTag or interfacing cable

3 Background Information

Multiplying (digital) circuit in real-world micro-controller and micro-processors use many algorithms, out of which the most efficient algorithm is the Booth's algorithm, in terms of the time for computation, simplicity of hardware architecture.

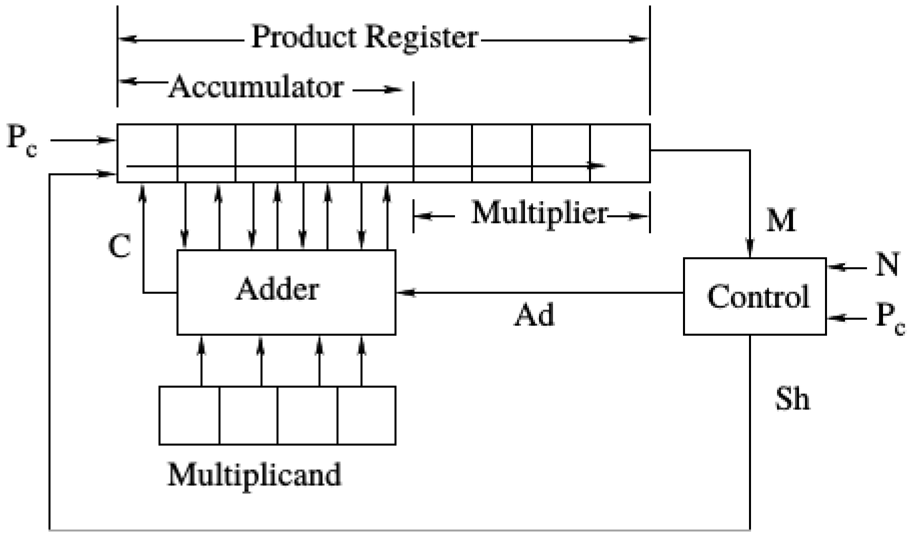
3.1 Serial-Parallel Multiplier Implementation

In this experiment, for academic reasons, we implement and then compare the serial-parallel multiplier with the Booth's algorithm. The former is a simple multiplier whose algorithm is same as the one you would do (since your school days), multiply the multiplicand by Least Significant Digit of multiplier, shift right, then repeat for the next digit to writeout 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.

The above idea is implemented in the circuit below. 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 coaded in your program. The results should appear on the LEDs on the FPGA board.

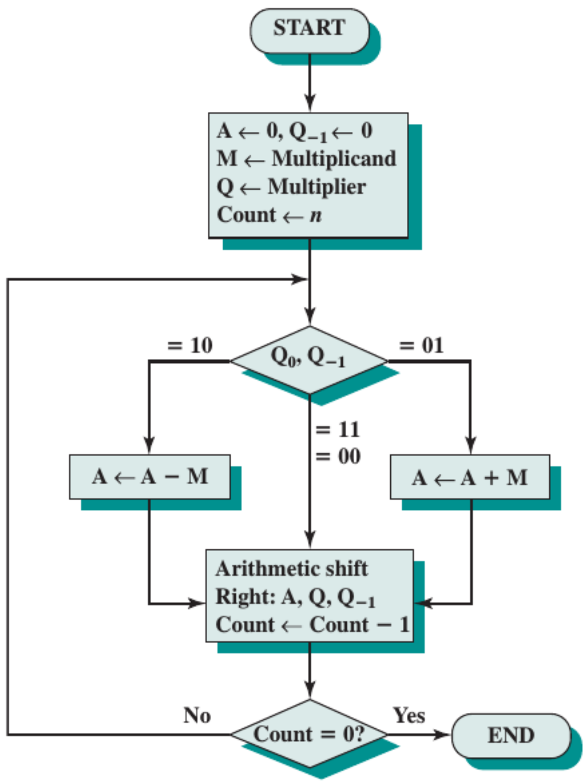
Figure 1 shows the multiplier to be implemented. The product register also serves as an accumulator to store the sum of the partial products. Note that the product register (content) is shifted to the right each time.

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.



3.2 Booth's 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.



4 Tasks to be Performed

4.1 Task 1:

Complete the blanks in the Verilog code below to describe the functionality of the unsigned multiplier.

```
module mult_4x4(

         input reset,clk,

         input.........A,B,

         output .......... O,      // 8–bits output

         output Finish);


reg [7:0] O;
```

```
wire Finish;
reg [3:0] State;                        // state machine
reg [8:0] ACC;                          // Accumulator
                        // logic to create 2 phase clocking when starting
assign Finish = (State==.......)? ..............;      // Finish Flag
always@(posedge clk, A,B)
begin
if(reset)
    begin
      State <= 0;
      ACC <= 0;
      O <= 0;
    end
else if (State==0)
        begin
        ACC[8:4] <= ...........;        // begin cycle
        ACC[3:0] <= ...........;        // Load A (one of our inputs)
        State <= 1;
        end

else if (State==1 || State == ..... || State ==5 || State == .....)
                                             // add/shift State

     begin
      if(ACC[0] == ............)
          begin                         // add multiplicand
         ACC[8:4] <= {1'b0,ACC[7:4]} + B;
         State <= State + 1;
          end
      else
```

```
            begin
                  ACC <= ...................;  // shift right
                  State <= State + 2;
            end
        end

else if(State==2 || State == ..... || State ==6 || State ==........)
                                                        // shift State

        begin
            ACC <= {1'b0,ACC[8:1]};      // shift right
            State <= State + 1;
        end

else if(State == ........... )
        begin
        State <= 0;
        O <= ..............;  // loading data of accumulator in output
        end
end
endmodule
```

4.2 Task 2:

Complete the testbench below and simulate the unsigned multiplier by giving appropriate clock signal

```
module test();
                                   // signals
reg reset,clk;
reg……. A,B;
                                   // Outputs
wire ………. O;
wire Finish;
                                   // device under test
mult_4x4 dut(reset, clk,A,B,O,Finish);
initial begin
reset=1;                           // reset
```

```
#40 A =……….; B= ………;
#10 $monitor ("%b",………….);
#400 reset = 0;
#40 start = 1;                    // start


$finish;
end

endmodule
```

4.3 Task 3:

Fill the blanks for the following code of booths algorithm

```
module multiplier(prod, busy, mc, mp, clk, start);
//mc is the mutiplicand & mp is the multiplier

output ………. prod;                 // ouput 8 bits
output busy;
input ……….. mc, mp;                // input 4 bits
input clk, start;
reg……….. 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 <= .....;
M <= mc;
Q <= mp;
Q_1 <= ….. ;  // bit written to the left of lsb of number to be multiplied
count <= 3'b0;
end

else
begin
case ({Q[0], Q_1})

……… : {A, Q, Q_1} <= {sum[3], sum, Q};
2'b1_0 : {A, Q, Q_1} <= {difference[3], difference, Q};
default: {……………} <= {A[3], A, Q};
endcase

count <= count + 1'b1;
end
```

```
end

alu adder(....., ..... ,  ..... , .....); // adder
alu subtracter(.... , .... , ...., ....); //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 ……… a;
input ……… b;
input cin;
assign out = a + b + cin;
endmodule
```

4.4 Task 4:

Write the testbench for the booths algorithm & simulate it

4.5 Task 5:

Write a \.ucf" file, implement the design on the FPGA board (hardcoding the input values) and show the output on the LEDs.

4.6 Comparison

Compare the performance of both these algorithms. <Hint: Track the number CPU cycles required for computation from Xilinx software>