Mup Assignment-1

Deepak Charan S ee23b022

August 2024

1 Introduction:

I am Deepak Charan S (Roll No: EE23B022) and this is my report for the first assignment of Microprocessor Lab, which I had done on 19/8/24.

2 Objective:

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

3 Equipments/Software Required:

1. The EDGE Artix 7 board (pic put karo)
2. PC hosting the Xilinx Vivado 2022 Software and USB interfacing cable

4 Procedure:

4.1 Serial Parallel Multiplier

In this way, we fed one operand in parallel (multiplicand) while another serially (multiplier) and had an accumulator which stores the repeated additions.
For every bit that comes serially from the multiplier, we either just shift right (if bit is 0) or add multiplicand to accumulator value and shift right (if bit is 1)

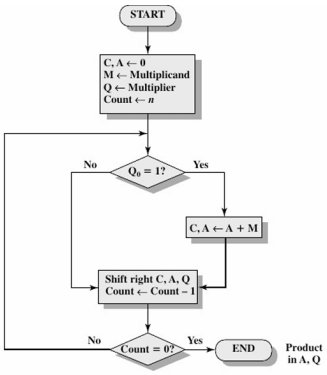
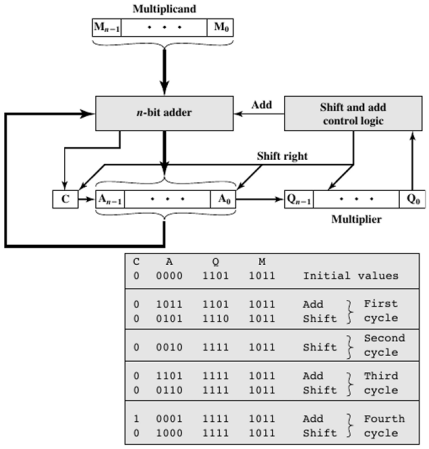


Figure 1: Flowchart



C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First
0	0101	1110	1011	Shift	cycle
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third
0	0110	1111	1011	Shift	cycle
1	0001	1111	1011	Add	Fourth
0	1000	1111	1011	Shift	cycle

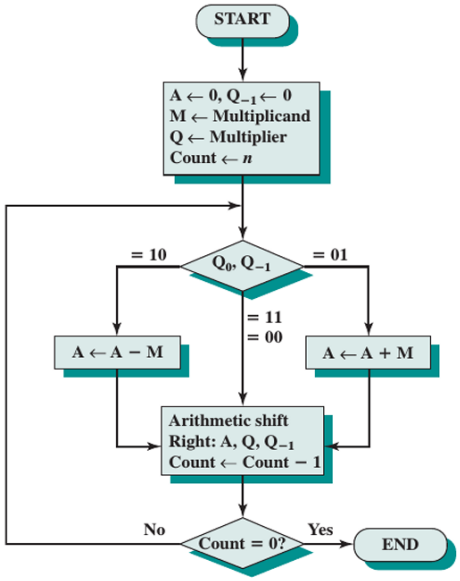
Figure 2: Example

4.2 Booth's Multiplier

In this case, we store -(multipicand) (2's complement of it), the previous bit accessed from multiplier and also an accumulator to store the repeated addition

Booths Multiplier

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First
1100	1001	1	0111	Shift	cycle
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third
0010	1010	0	0111	Shift	cycle
0001	0101	0	0111	Shift	Fourth cycle



Since we perform addition for only a few cases (Q0,Q-1 = (10)/(01)), we can greatly reduce the clock cycles needed to perform a multplication. (Number of times we shift remains same as SP Multiplier).

5 Codes:

5.1 Serial-Parallel Multiplier

Module

```
module sp_multiplier(
        input reset, clk,
        input [3:0]A, B,
        output reg [7:0] out, // 8-bits output
        output Finish);

    wire Finish;        //To denote end of multiplication
    reg [3:0] State; // state machine
    reg [8:0] ACC; // Accumulator

    // logic to create 2 phase clocking when starting
    assign Finish = (State==9) ? 1:0; // Finish Flag
    always@(posedge clk)
    begin
        if(reset)
        begin
            State <= 0;
            ACC <= 0;
            out <= 0;
        end
        else if (State==0)
        begin
            ACC[8:4] <= {1'b0,1'b0,1'b0,1'b0,1'b0}; // begin cycle
            ACC[3:0] <= A; // Load A (one of our inputs)
            State <= 1;
        end
        else if (State==1 || State == 3 || State ==5 || State == 7)
        begin // add/shift State
            if(ACC[0] == 1)
            begin // add multiplicand
                ACC[8:4] <= {1'b0,ACC[7:4]} + B;
                State <= State + 1;
            end
            else
            begin
                ACC <= {1'b0,ACC[8:1]}; // shift right
                State <= State + 2;
            end
        end
        else if(State==2 || State == 4 || State ==6 || State == 8)
        begin  // shift State
            ACC <= {1'b0,ACC[8:1]}; // shift right
            State <= State + 1;
        end
        else if(State == 9)
        begin
            State <= 0;
            out <= ACC[8:0]; // loading data of accumulator in output
        end
    end
endmodule
```

Test Bench

```
`timescale 1ns / 1ps

module tb_sp_multiplier();

    // signals
    reg reset,clk;
    reg [3:0] A,B;
    reg start;

    // Outputs
    wire [7:0] out;
    wire Finish;


    // device under test
    sp_multiplier dut(reset, clk, A, B, out, Finish);
    initial begin
        clk=0;
        reset = 1; // reset
        start = 1;
        #40 A = 5; B = 2;
        //$dumpfile("test_circuit.vcd");        //Used this code to simulate waveform in gtkwave on personal computer
        //$dumpvars(0,tb_sp_multiplier);
        #10 reset=0;
        #10 $monitor ("%b",out);
        #400 reset = 0;
        #40 start = 0; // start
        $finish;
    end

    always              // clock for simulation purposes
        begin

            #5 clk=~(clk);
        end
endmodule
```

5.2 Booth's Multiplier

Module

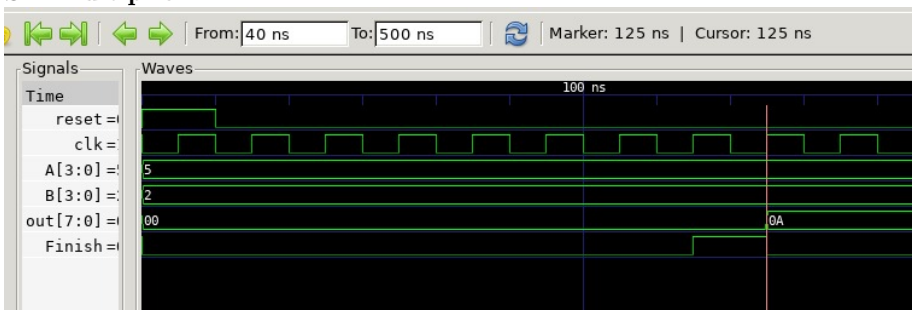
```
module booths_multiplier(out, busy, mc, mp, clk, start);

//mc is the mutiplicand & mp is the multiplier
output wire [7:0] out; // ouput 8 bits
output busy;
input [3:0] mc, mp; // input 4 bits
input clk, 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 <= 4'b0;
        M <= mc;
        Q <= mp;
        Q_1 <= 0; // bit written to the left of lsb of number to be multiplied
        count <= 3'b0;
    end
    else if(busy)
    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

alu adder(sum, A, M, 1'b0); // adder
alu subtracter(difference , A, (~M), 1'b1); //subtractor using 2's compliment
assign out = {A, Q}; // make it fill up the arguments
assign busy = (count < 4);
endmodule

// The following is an alu.It is an adder, but capable of subtraction:
// 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

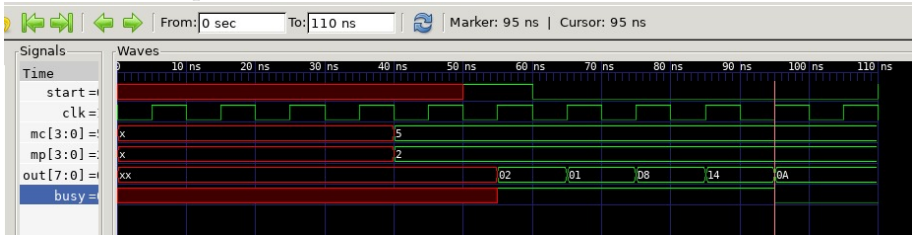
```
`timescale 1ns / 1ps


module tb_booths_multiplier();

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

    // Outputs
    wire [7:0]out;
    wire busy;

    // device under test
    booths_multiplier dut(out, busy, mc, mp, clk, start);
    initial begin
        $dumpfile("test_circuit.vcd");  //Used this code to simulate waveform in gtkwave on personal computer
        $dumpvars(0,tb_booths_multiplier);
        #40 mc = -3; mp = -2;
        #10 start=1;
        #10 start=0;
        //#10 $monitor ("%b, %b, %b",prod,mc,mp);

        #10 $monitor ("%b",out);
        #40 start = 1;
        $finish;
    end

    initial
        begin
            clk <= 0;
            forever #5 clk <=~(clk);
        end
endmodule
```

6 Waveforms:

SP Multiplier



Booth's Multiplier



7 Result:

- Analysing the waveforms, we see that SP multiplier took 7 cycles while Booth's took only 3; making it a far more superior algorithm to implement.
- Booth's can also perform multiplication on signed binary numbers, making it more versatile.
- In terms of contribution, I had taken upon the Booth's Algorithm part (writing verilog code, its testbench and generating its waveform)

8 References:

The Handouts and User Manual given in Moodle