CS2610: Computer Organization and Architecture Lab 4

Problem Statement:

Design a 32 bit ALU which can perform 32-bit signed addition/subtraction and multiplication. You are required to use look-ahead Adder and Booth's Multiplier for this ALU. The ALU has two 32 bit inputs (operands), one control input (17 bits used as opcodes), and one output which is a combination of two 32 bit registers (UH and LH). LH is lower half and it contains 0-31 bits of the output, while UH is upper half which holds bit 32-63 of the output. In this lab, you should be able to perform ADD, SUB and MUL operations for which opcodes are given in the table below.

Funct 7	Func3	Opcode
1000000	000	0110011
0100000	000	0110011
0000001	000	0110011

Please design the ALU circuit and also perform extensive testing with your very own test bench

Code and it’s eplaination::

We have declared ALU module inamed "look" and has 5 inputs and outputs. The inputs are two 4-bit numbers (a and b), a carry-in bit (cin), and the outputs are a 4-bit sum (sum) and a carry-out bit (cout). The module implements the logic for adding or subtracting two 4-bit numbers using a "look-ahead" carry generator. The output sum is computed by XOR-ing the input numbers and the carry bits, while the carry-out is computed using logic gates and the carry bits.

The "over" module takes three inputs (a, b, and c) and has one output (overf). It checks if the inputs satisfy the condition for overflow in subtraction (a = b ≠ c) and sets the output to 1 if the condition is true, and 0 otherwise.

The "ADDSUB" module takes three inputs (A, B, and op) and has three outputs (sum, co, and overflow). A and B are 32-bit numbers, op is a control signal that determines whether to add or subtract the numbers. The module first selects the correct input (A or B) based on the value of op, and then splits the input numbers into 8 4-bit blocks that are processed by the "look" module. The carry-out from the last block is assigned to the output co, and the 32-bit sum is concatenated with zeros to form the 64-bit output. The "over" module is used to check for overflow in subtraction and set the overflow output accordingly.

The "MUL" module takes two signed 32-bit numbers (A and B) as inputs and produces a signed 64-bit number (C) as output. The module implements a shift-and-add algorithm for multiplication. It initializes C to 0 and iteratively multiplies A with each bit of B, shifting the partial product by 1 bit to the left each time. If the bit of B is 1, the partial product is added to C, otherwise it is discarded. The module also checks if A is -8 (a special case) and negates C if true.

```
module look(input [3:0] a,    input [3:0] b,
input cin,
output [3:0] sum,
output cout    );
wire [3:0] p,g;
wire [3:0]c;
assign p=a^b;
assign g=a&b;
assign c[0]=cin;
 assign c[1]=g[0]|(p[0]&cin);    assignc[2]=g[1]|(p[1]&g[0])|(p[1]&p[0]&cin);    .
assign c[3]=g[2]|(p[2]&g[1])|(p[2]&p[1]&g[0])|(p[2]&p[1]&p[0]&cin);
Assign cout=g[3]|(p[3]&g[2])|(p[3]&p[2]&g[1])|(p[3]&p[2]&p[1]&g[0])|(p[3]&p[2]&p[1]&p[0]&cin);
assign sum=p^c;
Endmodule
module over(    input a,    input b,    input c,    output reg overf    );
   always @(a or b or c)
              if ((a==b) && (a!=c)) begin
                            overf<=1;
              end
              else
                            overf<=0;
endmodule
```

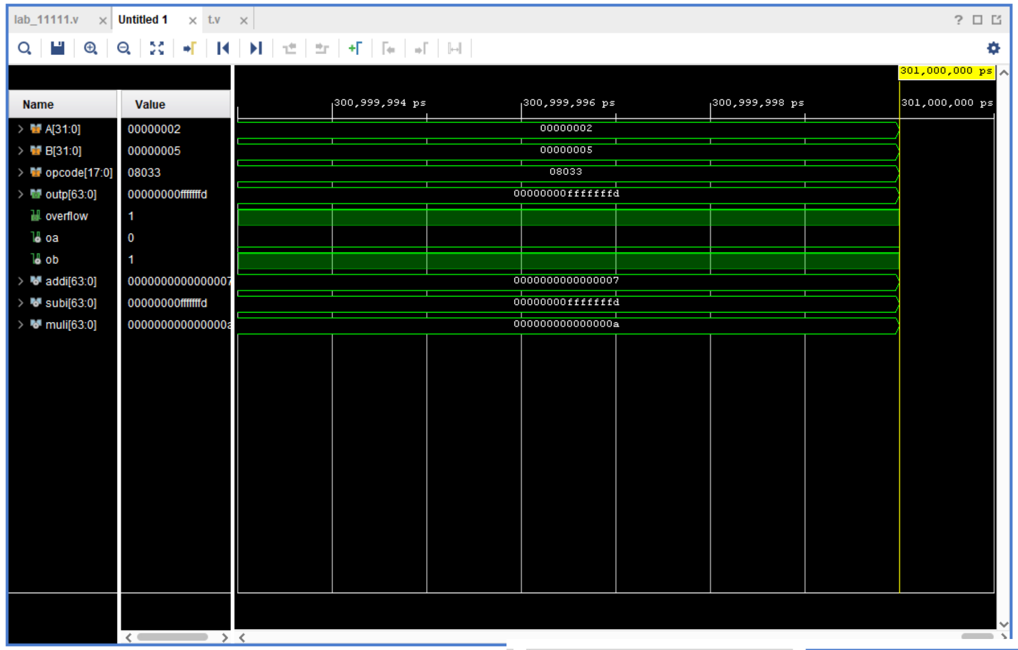
```
module ADDSUB(    input [31:0]A,    input [31:0]B,    input op,    output [63:0]sum,
output co,    output overflow    );
wire [31:0]Bi,sel;
 assign sel=op? Bi:B;
assign Bi=~B+1;
 wire [31:0]s;
 wire c1,c2,c3,c4,c5,c6,c7,c8;
                look l1(.a(A[3:0]),.b(sel[3:0]),.cin(1'b0),.sum(s[3:0]),.cout(c1));
                look l2(.a(A[7:4]),.b(sel[7:4]),.cin(c1),.sum(s[7:4]),.cout(c2));
                look l3(.a(A[11:8]),.b(sel[11:8]),.cin(c2),.sum(s[11:8]),.cout(c3));
                look l4(.a(A[15:12]),.b(sel[15:12]),.cin(c3),.sum(s[15:12]),.cout(c4));
                look l5(.a(A[19:16]),.b(sel[19:16]),.cin(c4),.sum(s[19:16]),.cout(c5));
                look l6(.a(A[23:20]),.b(sel[23:20]),.cin(c5),.sum(s[23:20]),.cout(c6));
                look l7(.a(A[27:24]),.b(sel[27:24]),.cin(c6),.sum(s[27:24]),.cout(c7));
                look l8(.a(A[31:28]),.b(sel[31:28]),.cin(c7),.sum(s[31:28]),.cout(c8));
 assign co=c8;
 assign sum={32'd0,s};
over o(.a(A[31]),.b(B[31]),.c(sum[31]),.overf(overflow));
endmodule
```

```
module MUL(    input signed [31:0]A,input
signed [31:0]B,    output reg signed [63:0]C
);
reg [1:0]temp;
reg E1;
reg [31:0] A1;
integer i;
always @ (A, B)
                begin
                 C=64'd0;
                E1=1'd0;
for ( i = 0 ; i < 32 ; i=i+1)        begin
                temp={B[i],E1};
                A1=-A;
 case (temp)
                2'd2: C[63:32]=C[63:32]+A1;
                2'd1: C[63:32]=C[63:32]+A;
default: begin end
endcase
                C=C>>1;
                C[63]=C[62];
                E1=B[i];
                End
if (A==32'd8)
begin           C=-C;
```

```
end
endendmodule
module ALU(    input [31:0]A,    input [31:0]B,    input [17:0]opcode,
output reg [63:0]outp,    output reg overflow    );
 wire oa,ob;
wire [63:0]addi,subi,muli;
MUL m1(.A(A),.B(B),.C(muli));
ADDSUB a1(.A(A),.B(B),.op(1'b0),.sum(addi),.overflow(oa));
ADDSUB s1(.A(A),.B(B),.op(1'b1),.sum(subi),.overflow(ob));
always @(A or B or opcode)    begin
                case (opcode)
                                17'b10000000000110011: begin
                                                outp<=addi;
                                                overflow<=oa;
                                end
                                17'b01000000000110011: begin
                                                outp<=subi;
                                                overflow<=ob;
                                end
                                17'b00000010000110011: outp<=muli;
                                default: begin end
                endcase
        end
endmodule
```

Testbench::

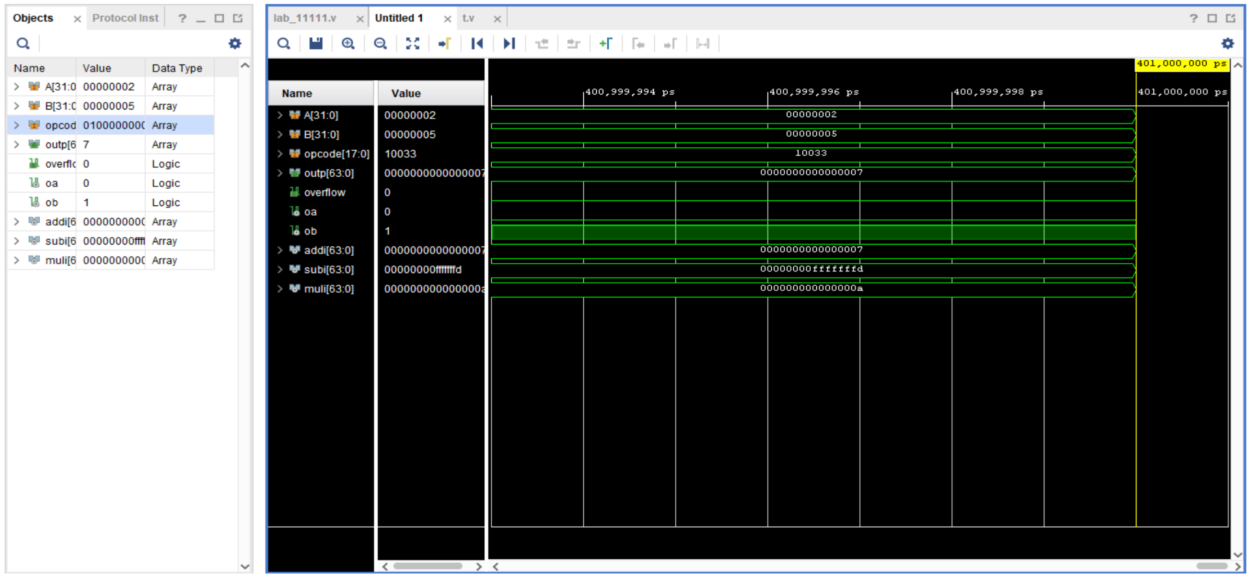
```
module ALUtest;
reg [31:0]A;
reg [31:0]B;
reg [16:0]opcode;
wire [63:0]outp;
wire overflow;
        ALU dut(.A(A),.B(B),.opcode(opcode),.outp(outp),.overflow(overflow));
        Initial
         begin
                A=34;
                 B=23;
                opcode=17'b10000000000110011;
        if (overflow==1'b1) $display("Overflow occured in addition");
                 #40;
                 A=23;
                B=11;
                opcode=17'b01000000000110011;
        if (overflow==1'b1) $display("Overflow occured in subtraction");
        #40;
         A=62;
        B=-4;
        opcode=17'b00000010000110011;
        #40;
  endendmodule
```


lab_11111.v | Untitled 1 | t.v

301,000,000 ps

300,999,994 ps | 300,999,996 ps | 300,999,998 ps | 301,000,000 ps

Name	Value
A[31:0]	00000002
B[31:0]	00000005
opcode[17:0]	08033
outp[63:0]	00000000fffffffd
overflow	1
oa	0
ob	1
addi[63:0]	0000000000000007
subi[63:0]	00000000fffffffd
muli[63:0]	000000000000000a


Objects | Protocol Inst

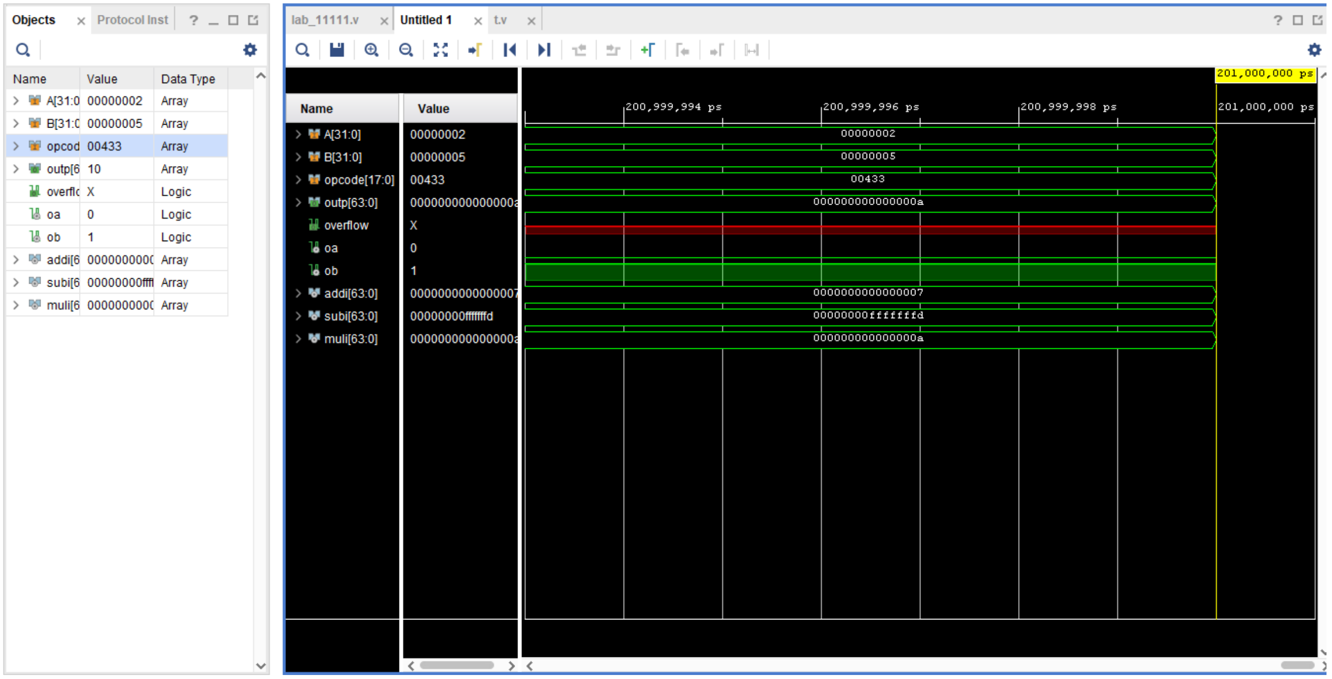
Name	Value	Data Type
A[31:0	00000002	Array
B[31:0	00000005	Array
opcod	010000000(	Array
outp[6	7	Array
overflc	0	Logic
oa	0	Logic
ob	1	Logic
addi[6	000000000(	Array
subi[6	00000000fffl	Array
muli[6	000000000(	Array

lab_11111.v | Untitled 1 | t.v

401,000,000 ps

400,999,994 ps | 400,999,996 ps | 400,999,998 ps | 401,000,000 ps

Name	Value
A[31:0]	00000002
B[31:0]	00000005
opcode[17:0]	10033
outp[63:0]	0000000000000007
overflow	0
oa	0
ob	1
addi[63:0]	0000000000000007
subi[63:0]	00000000fffffffd
muli[63:0]	000000000000000a


Objects | Protocol Inst

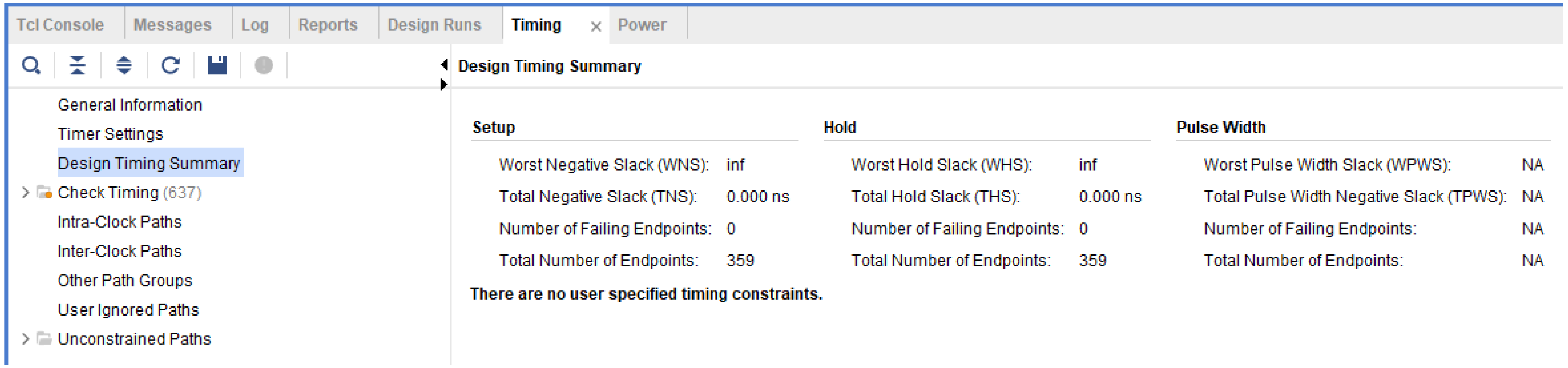
Name	Value	Data Type
A[31:0	00000002	Array
B[31:0	00000005	Array
opcod	00433	Array
outp[6	10	Array
overflc	X	Logic
oa	0	Logic
ob	1	Logic
addi[6	000000000(	Array
subi[6	00000000fffl	Array
muli[6	000000000(	Array

lab_11111.v | Untitled 1 | t.v

201,000,000 ps

200,999,994 ps | 200,999,996 ps | 200,999,998 ps | 201,000,000 ps

Name	Value
A[31:0]	00000002
B[31:0]	00000005
opcode[17:0]	00433
outp[63:0]	000000000000000a
overflow	X
oa	0
ob	1
addi[63:0]	0000000000000007
subi[63:0]	00000000fffffffd
muli[63:0]	000000000000000a



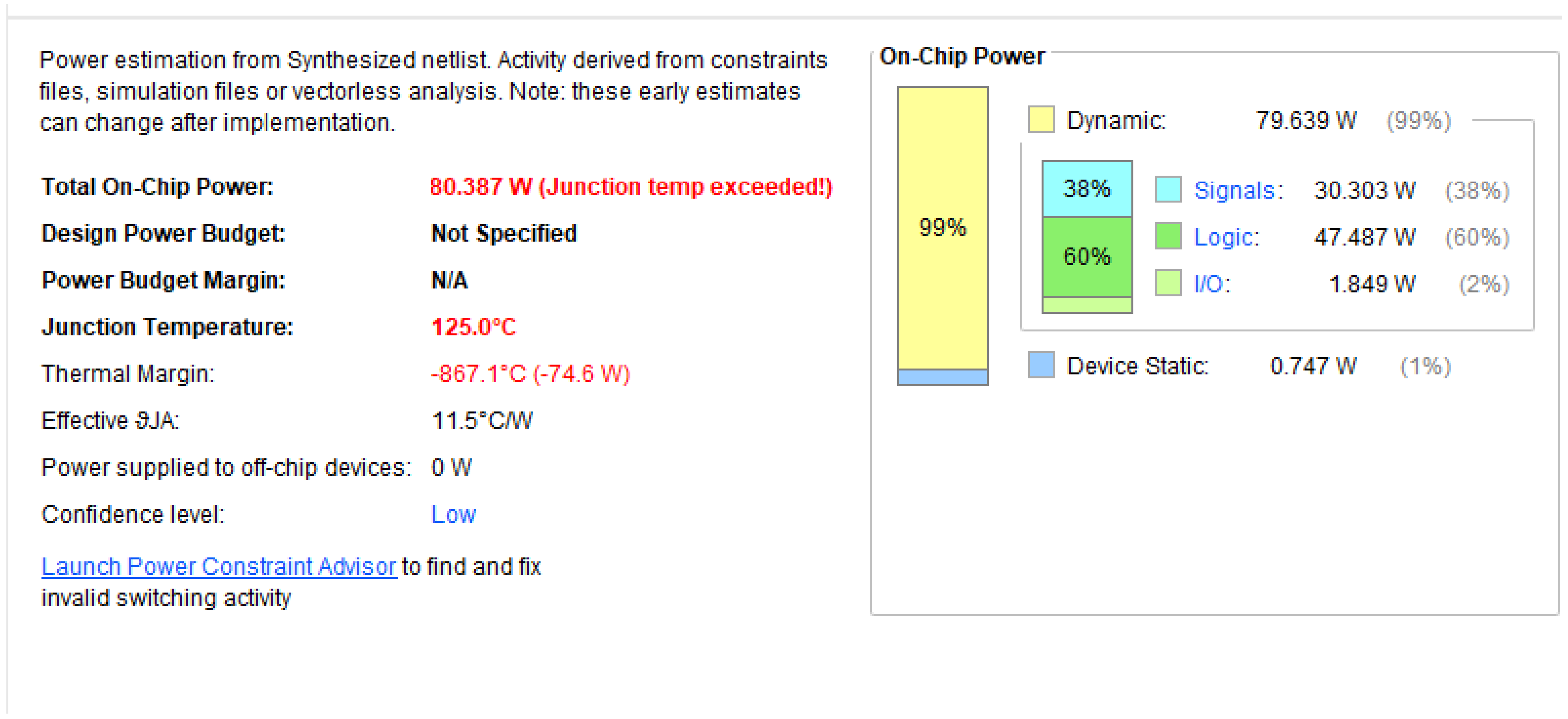
Tcl Console | Messages | Log | Reports | Design Runs | Timing | Power

- General Information
- Timer Settings
- Design Timing Summary
- Check Timing (637)
- Intra-Clock Paths
- Inter-Clock Paths
- Other Path Groups
- User Ignored Paths
- Unconstrained Paths

Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	359	Total Number of Endpoints:	359	Total Number of Endpoints:	NA

There are no user specified timing constraints.



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**80.387 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-867.1°C (-74.6 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	79.639 W	(99%)
Signals:	30.303 W	(38%)
Logic:	47.487 W	(60%)
I/O:	1.849 W	(2%)
Device Static:	0.747 W	(1%)

Conclusion:

The look-ahead adder is a fast adder that reduces the propagation delay of the carry signal by generating carry signals for all possible bit positions simultaneously. The Booth's multiplier is an algorithm that reduces the number of partial products generated during multiplication by using a signed digit representation of the multiplier. It generates partial products only when there is a transition from 0 to 1 or 1 to 0 in the multiplier. The partial products are then added using a multi-precision adder to generate the final result.

Contribution::
112101035_Jignesh::Verilog code
112101030_ Prakash:: Testbench
112101013_ Rakesh:: lab report