DIGITAL ASSIGNMENT

OF

EEE F313 ANALOG & DIGITAL VLSI DESIGN

Project done by:

Shantanu Nigam	2017A8PS0399P
Sarthak Mahapatra	2017A8PS0456P
Deepansh Goyal	2017A3PS0312P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

CPL is a very important technology used in digital electronics. It reduces the number of transistors required to produce outputs. It also uses restoring transistors for pull up logic. This takes complimentary inputs and generates complimentary outputs. Similar technology is also used in SRAM design which is used in cache architecture.
In part B we have made CLA adder. A Carry look ahead adder generates output faster than a normal adder. For 4 bit inputs, it produces output in just 1 clock pulse where as a normal adder requires 4 clock pulse. This though comes at the cost of greater architectural complexity.

PROBLEM STATEMENT

Part A

Logic Function Realization

F= AB +A'C' +AB'C '

Design of a 3-variable Boolean function using CPL using minimum transistors at 500MHz with load capacitance of 1pF.

Part B

Design a simple four-bit carry-look- ahead (CLA) adder using gate level modelling and cascade two such stages in ripple carry fashion to build an eight-bit adder. Use only two input NAND gates in the whole design.

Technology Used

- MOSFETs in SCL 018 technology.
- Core voltage (VDD) for this technology is 1.8V.
- Model file (contains both NMOS and PMOS transistor)

PART A

F = AB +A'C' +AB'C '

The Boolean expression was reduced to = (A AND B) OR (A XNOR C)

Approach to Problem

In CPL technology, to implement the schematic level we need to calculate design specifications (W/L) for CPL circuit part and output level restoring PMOS part. The output capacitance was given as 1pF, which we were unable to drive at 500MHz. So we took reduced capacitance value to be 100fF.

In CPL we have assumed that we have complimentary inputs available with us. This assumption is valid in the case of an actual chip using CPL. The previous and the next stage all are available in real and complimentary values. Then this assumption is valid.

Table-1 - W/L values for different transistors.

Stage of MOSFET	W/L value
First stage in CPL pass transistor NMOS	3.6u/180n
Second stage in CPL pass transistor NMOS	5u/180n
Restoring NMOS transistor	1.2u/180n

For CPL circuit part

Initially we tried sweeping using different values of W for different stages of the logic circuit. We observed that as we increased the W value for the next stage, previous stage output will degrade and could be improved only by increasing W value for the previous stage. Thus by various tries we got to some finite values for W/L.

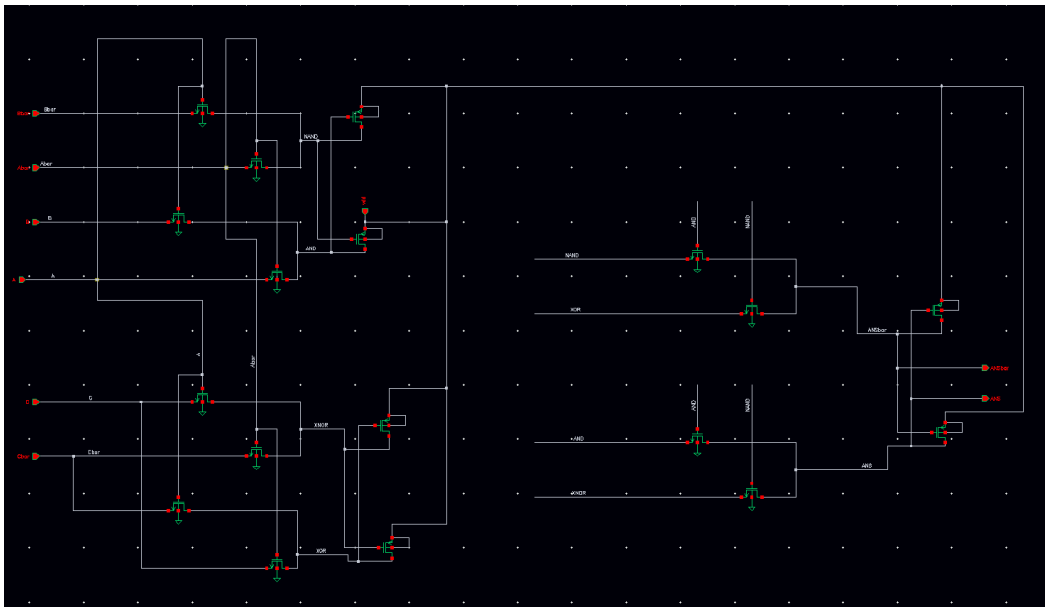

Figure 12 Full Schematic

For PMOS part

It is known that NMOS is a weak HIGH logic transfer gate. So, when the Pass Transistor has to pass HIGH logic, the output is not Vdd but Vdd-Vth_N. To get the full output swing in the output we have used logic restoring PMOS, which will give charging current to the capacitor when logic high appears. The W/L value for this PMOS should not be very high otherwise it will prevent to restore logic LOW.

To get W/L value for PMOS transient analysis was done at different values and finally W/L=1200n/180n gave full output voltage in minimum time.

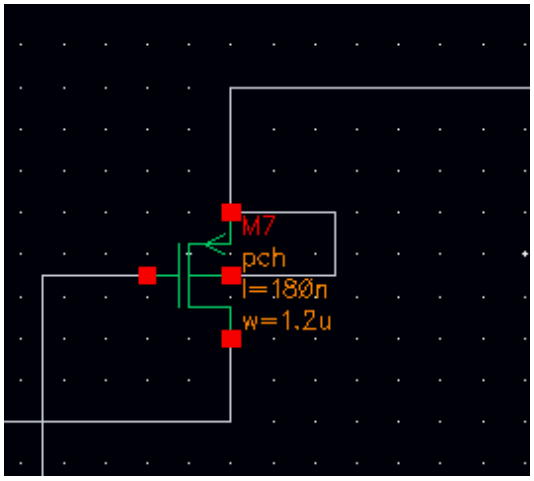

Figure 13: W = 1.2 micrometres for PMOS.

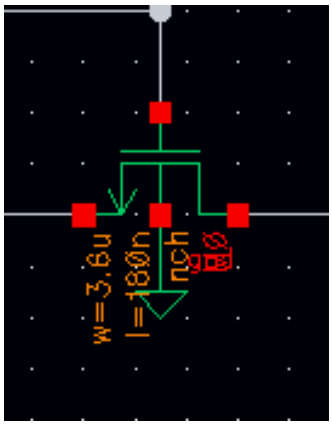

Figure 14: First stage Pass Transistor NMOS

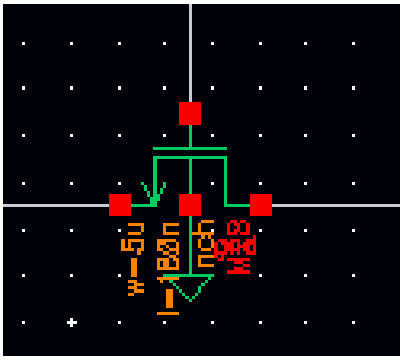

Figure 15: Second stage Pass Transistor NMOS

Layout:

In layout for NMOS, we have used multi-fingered gate layout. This helped us to reduce parasitic capacitance between MOSFETs. Moreover, we have taken a common drain between 2 NMOS in first stage to reduce wire capacitance and delay. This was however not possible in second stage because we had 4 fingered gate giving same terminal on both the ends of the MOSFET. So, wires had to be used for connection.

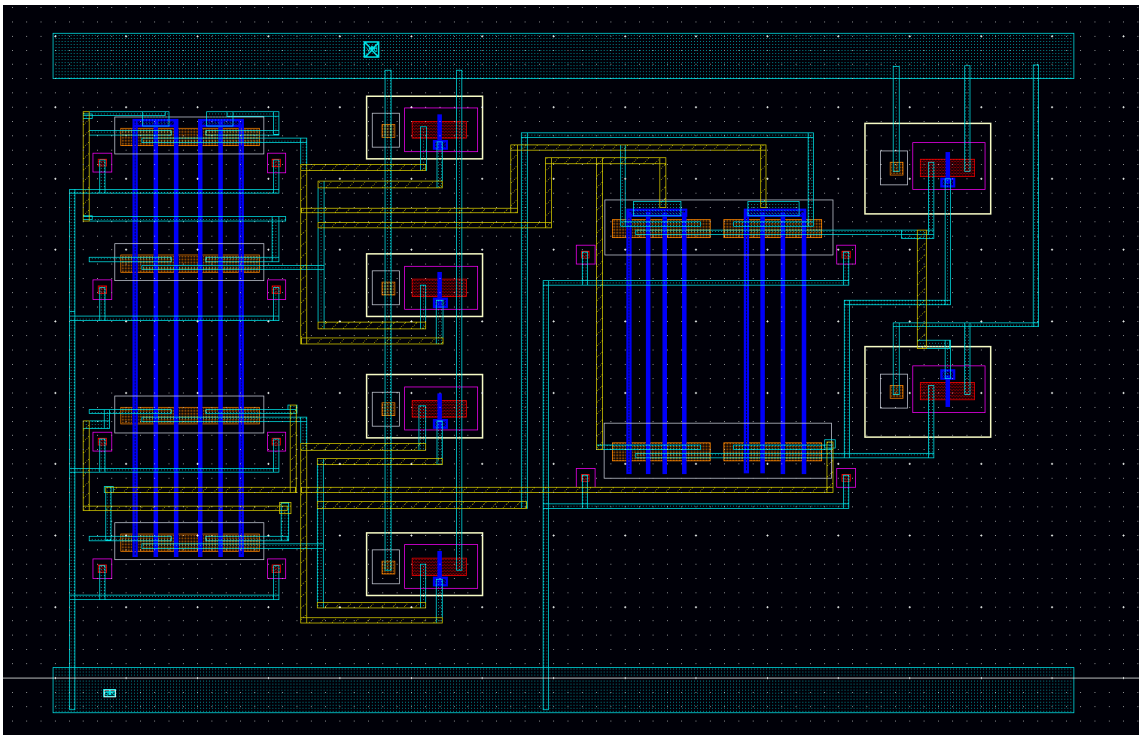

Figure 16: Full Layout

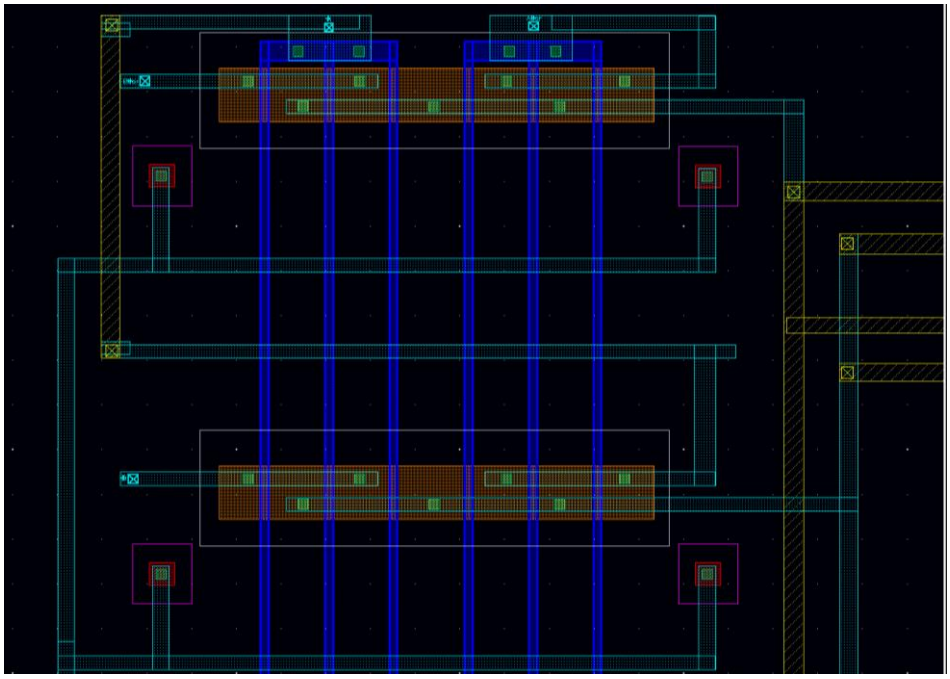

Figure 17: First stage NMOS for AND-NAND complimentary pair implementation

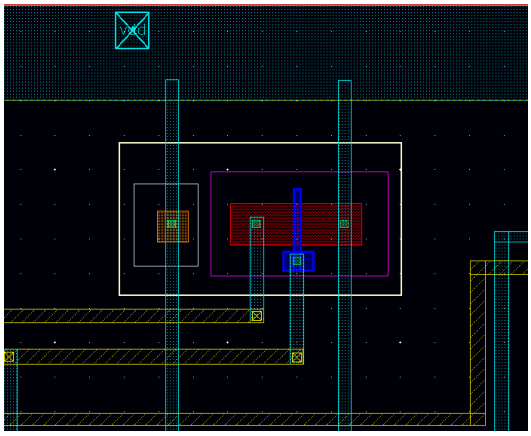

Figure 18: Restoring PMOS

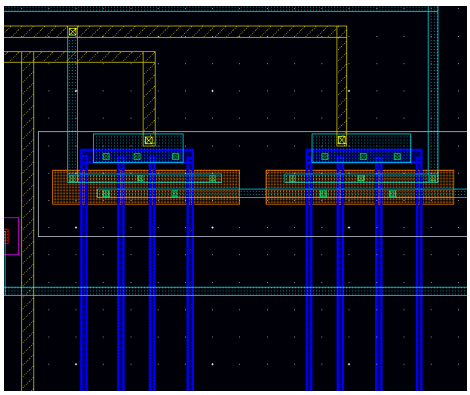

Figure 19: Second stage NMOS with 4 fingered-gates

DRC and LVS was successfully passed for the schematic and PEX extracted file was used to do simulations. The results are as follows.

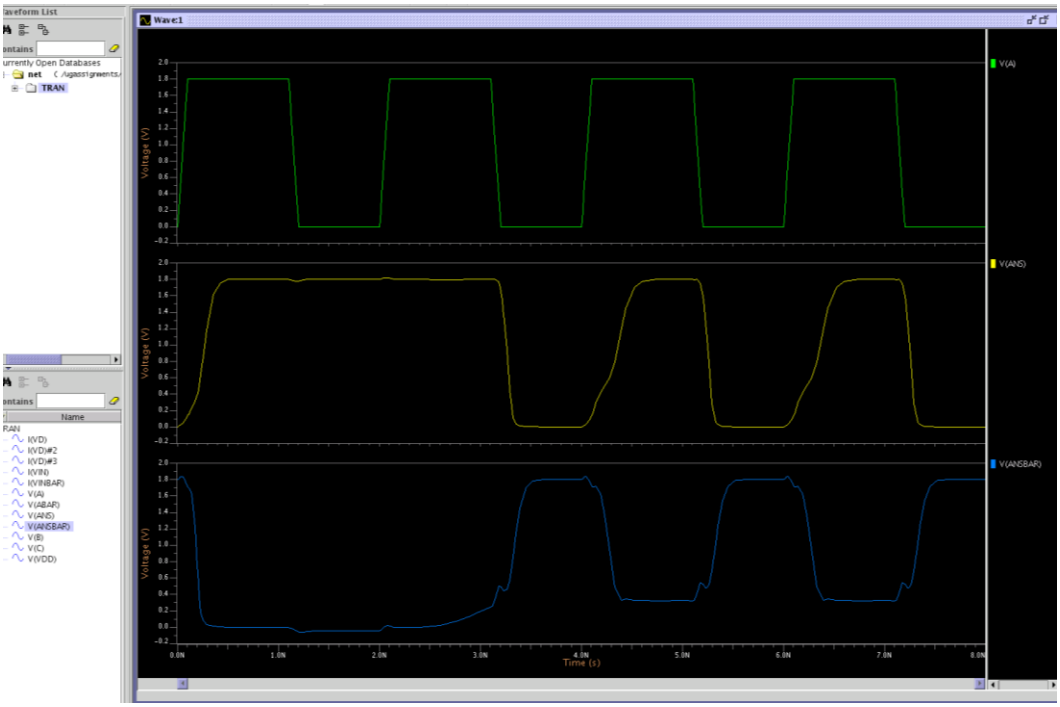

Figure 20: ANS and ANSbar plotted against A when other outputs were high

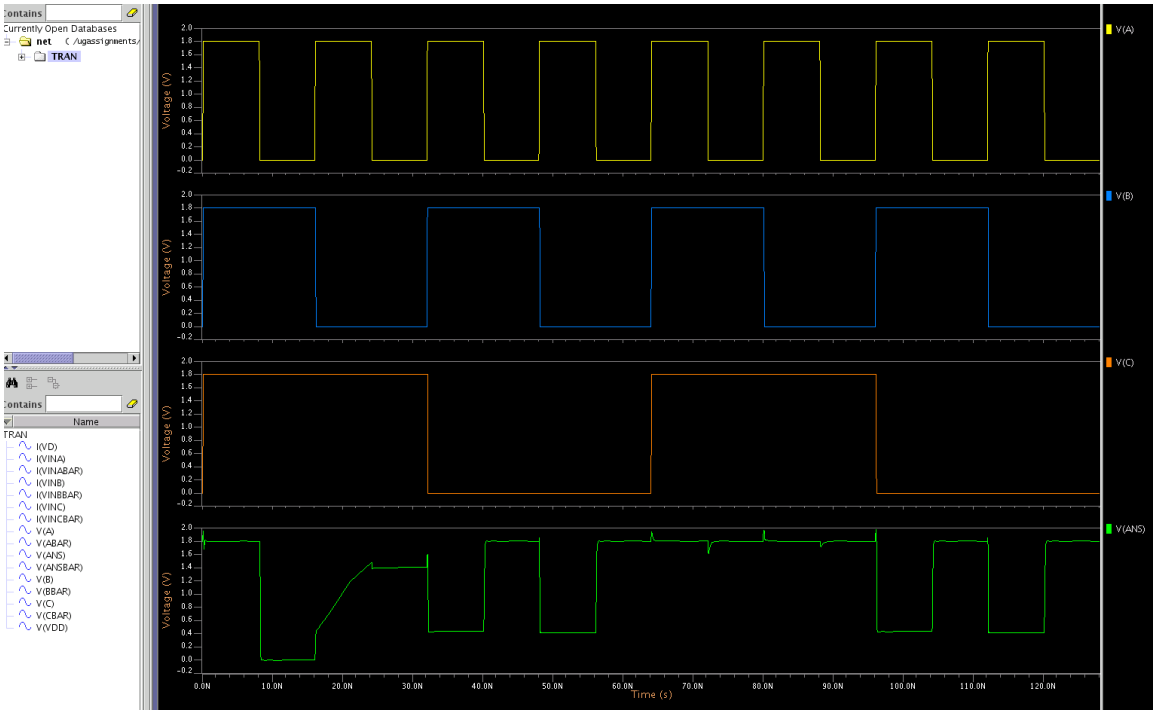

Figure 21: Testing against all inputs

Part B

We have used gate level modelling in Verilog and made a full adder module.
Using the full adder, we made a 4 bit Carry Look Ahead Adder. In this, only NAND gates were used.
Finally, two modules of 4-bits adders were combined together to form an 8-bit adder.

The file includes a testbench as well.

The code is as follows:

```
module fulladder (a, b, c, sum, cout);


        input  a;

        input  b;

        input  c;

        output sum;

        output cout;


        wire   w1;

        wire   w2;

        wire   w3;

        wire   w4;

        wire   w5;

        wire   w6;

        wire   w7;


        nand( w1, a , b);

        nand( w2, w1, a);

        nand( w3, w1, b);

        nand( w4, w2, w3);

        nand( w5, w4, c);
```

```
    nand( w6, w5, w4);
    nand( w7, w5, c);
    nand( sum, w6, w7);
    nand(cout, w5, w1);

endmodule // full_adder

module claAdder(input [3:0]  i1, input [3:0]  i2, input cin , output [3:0] out, output cout);

        wire [3:0] wG; //generate term
        wire [3:0] wP; //propagate term
        wire [3:0] wC; //stores the carry out of the full adder < not required as we are using CLA design>
        wire [2:0] C; // direct comuputed c values
        wire [3:0] wNAND; // A nand B
        wire [3:0] waab; //a nand (a nand b)
        wire [3:0] wbab; //b nand (a nand b)
        wire [3:0] wPC;

        fulladder fa0(i1[0] , i2[0], cin, out[0], wC[0]);

        nand( wNAND[0], i1[0], i2[0]);
        nand( wG[0], wNAND[0], wNAND[0]);
        nand(waab[0], wNAND[0], i1[0]);
        nand(wbab[0], wNAND[0], i2[0]);
        nand(wP[0], waab[0], wbab[0]);
        nand(wPC[0], wP[0], cin);
        nand(C[0], wNAND[0], wPC[0]);
```

```
fulladder fa1(i1[1] , i2[1], C[0], out[1], wC[1]);



nand( wNAND[1], i1[1], i2[1]);

nand( wG[1], wNAND[1], wNAND[1]);

nand(waab[1], wNAND[1], i1[1]);

nand(wbab[1], wNAND[1], i2[1]);

nand(wP[1], waab[1], wbab[1]);

nand(wPC[1], wP[1], C[0]);

nand(C[1], wNAND[1], wPC[1]);

fulladder fa2(i1[2] , i2[2], C[1], out[2], wC[2]);

nand( wNAND[2], i1[2], i2[2]);

nand( wG[2], wNAND[2], wNAND[2]);

nand(waab[2], wNAND[2], i1[2]);

nand(wbab[2], wNAND[2], i2[2]);

nand(wP[2], waab[2], wbab[2]);

nand(wPC[2], wP[2], C[1]);

nand(C[2], wNAND[2], wPC[2]);

fulladder fa3(i1[3] , i2[3], C[2], out[3], wC[3]);

nand( wNAND[3], i1[3], i2[3]);

nand( wG[3], wNAND[3], wNAND[3]);

nand(waab[3], wNAND[3], i1[3]);
```

```
        nand(wbab[3], wNAND[3], i2[3]);
        nand(wP[3], waab[3], wbab[3]);
        nand(wPC[3], wP[3], C[2]);
        nand(cout, wNAND[3], wPC[3]);


endmodule

module eightbitadder(input [7:0]  a, input [7:0]  b, input cin , output [7:0] sum, output cout);

        wire cbw;

        claAdder cla1 (a[3:0],b[3:0],cin, sum[3:0], cbw);
        claAdder cla2 (a[7:4],b[7:4],cbw, sum[7:4], cout);

endmodule

module testbench;

        wire [7:0] sum;
        wire cout;
        reg [7:0]a;
        reg [7:0]b;
        reg cin;

        eightbitadder adder (a[7:0],b[7:0],cin, sum[7:0], cout);
```

```
initial
    begin
            a=8'b00000000; b=8'b00000000; cin=1'b0;
        #10 a=8'b00000000; b=8'b01001010; cin=1'b0;
        #10 a=8'b01110000; b=8'b00000000; cin=1'b0;
        #10 a=8'b10010011; b=8'b00000100; cin=1'b0;
        #10 a=8'b00011001; b=8'b00001100; cin=1'b0;
        #10 a=8'b01110010; b=8'b00110000; cin=1'b0;
        #10 a=8'b01010110; b=8'b00100110; cin=1'b0;
        #10 a=8'b01110010; b=8'b01000110; cin=1'b0;
        #10 a=8'b01001100; b=8'b00001010; cin=1'b0;
        #10 a=8'b00001001; b=8'b00000101; cin=1'b0;
        #10 a=8'b10100101; b=8'b01001010; cin=1'b0;
        #10 a=8'b11111111; b=8'b11111111; cin=1'b0;
        #10 a=8'b00000000; b=8'b01001010; cin=1'b1;
        #10 a=8'b01110000; b=8'b00000000; cin=1'b1;
        #10 a=8'b10010011; b=8'b00000100; cin=1'b1;
        #10 a=8'b00011001; b=8'b00001100; cin=1'b1;
        #10 a=8'b01110010; b=8'b00110000; cin=1'b1;
        #10 a=8'b01010110; b=8'b00100110; cin=1'b1;
        #10 a=8'b01110010; b=8'b01000110; cin=1'b1;
        #10 a=8'b01001100; b=8'b00001010; cin=1'b1;
        #10 a=8'b00001001; b=8'b00000101; cin=1'b1;
        #10 a=8'b10100101; b=8'b01001010; cin=1'b1;
        #10 a=8'b11111111; b=8'b11111111; cin=1'b1;
```

```
                    #250 $finish;

                end

        endmodule
```

Testbench results:

Figure 22.1: Testbench simulation

Figure 11.2: Testbench simulations