King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department
COE 360-Term031

Pipelined Parallel Multiplier Project
Phase II: Circuit Design

Prepared for:
Dr. Mohammed Elrabaa

By
Mohammed Rushdi Ahmed 208529 Sec#01
Asmat Khaled Marouf 208675 Sec#02

TABLE OF CONTENTS

I. PART ONE

INTRODUCTION

1.Project Description:

Design 6-bit pipelined parallel multiplier with a clock frequency of 800 MHz.

2.Purpose:

The aim of this phase of the project is to design the transistor level implementation of a 6-bit pipelined parallel multiplier, including some *SPICE* simulation results.

3.Constraints and requirements:

The main constraint of this project is to make the multiplier functions at a **frequency of 800MHz , with load capacitance of 2pF .** Moreover; to meet good design aspects, the following points should be satisfied:

1- Minimum number of transistors.
2- Smallest possible transistor sizes.
3- All path delays to be equal.
4- Symmetrical noise margins.
5- Symmetrical rise and fall delays.

4.Theory:

In order to meet the above conditions, we started with the following considerations:

1-for minimum number of transistors, we performed some logic transformation on some logic paths (as shall be indicated in the procedure) to achieve the CMOS logic form as possible.

2-in order to get symmetrical noise margins and symmetrical rise and fall delays, we considered **Rpu =Rpd.** Which implicates that **Wpeff =2.5Wneff**.

The parallel pipelined multiplier we designed consists of 4 pipeline stages with. Therefore , we must consider the longest path to be assigned the calculations that would produce **the worst case** of sizes, which ensures the best match to the requirements.

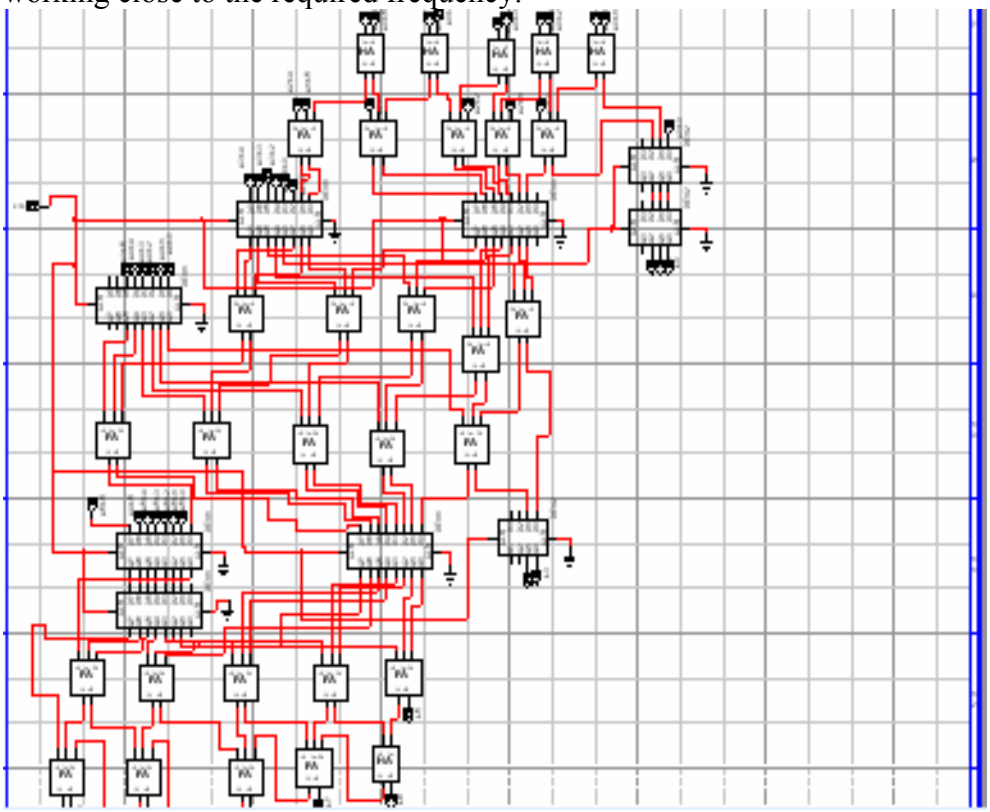
The theoretical results are still under-estimated. So, when simulating our circuit using *SPICE,* we expect to get results that are not very accurate and, as a result, we must go through tweaking and trials until we reach the best possible results.

5. Logic Design:

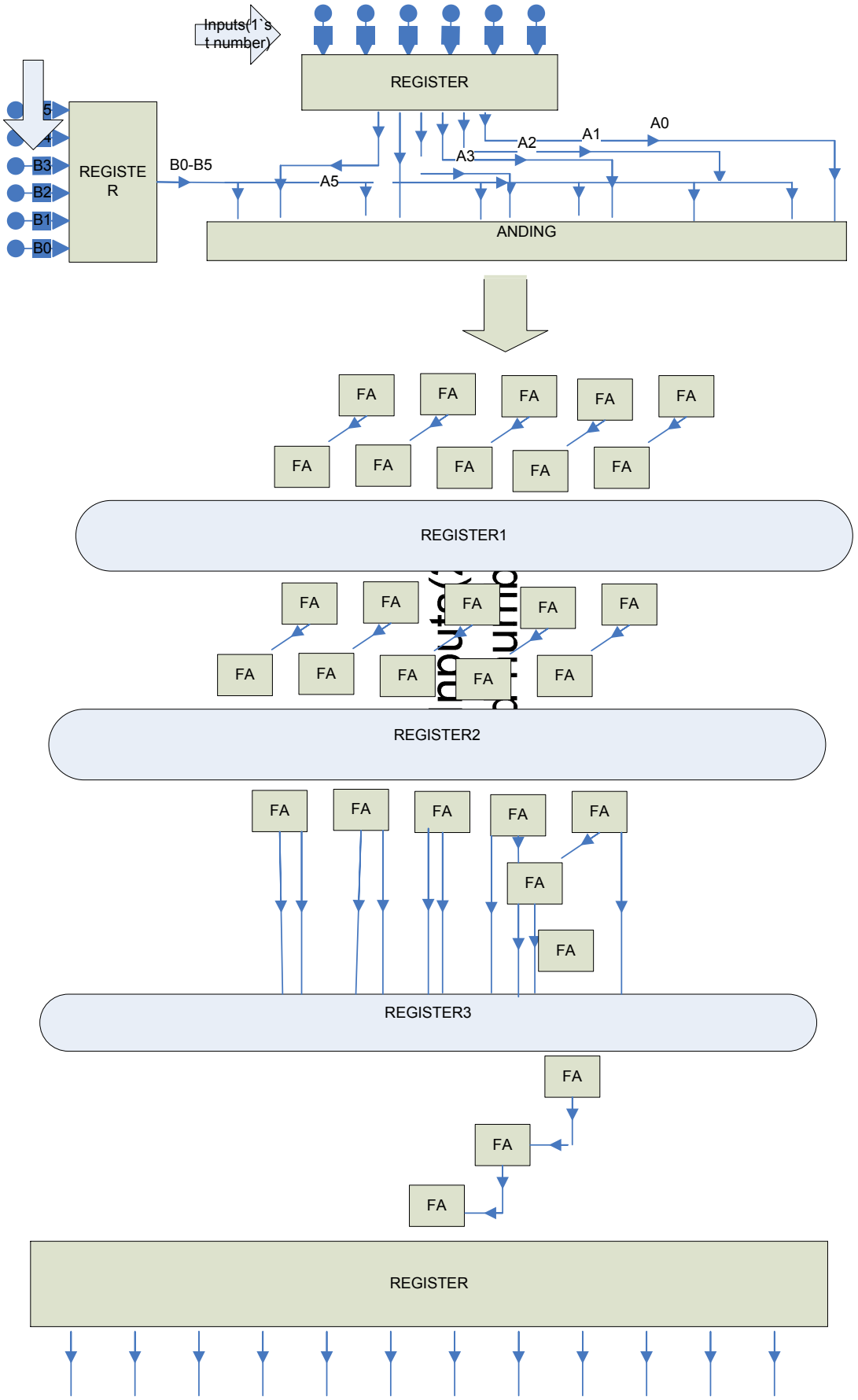
NOTE: the logic design that we designed in phase one was not based on parallel multiplication. Therefore, the requirement of making the multiplier to work at 800MHz would be too complicated and difficult to achieve. Thus, we **modified** our logic design to meet the requirement of the project. The block diagram of our new design is provided in the next page.

The new logic design consists of 4 pipeline stages.
The first2 stages contains 2 levels of full adders while the last 2 consist of 3 levels of full adders. Except the last stage and part of the 3rd stage, the carries resulting from every full adder is sent (diagonally) to another full adder in the lower level without waiting the carry of the previous full adder of that level.

We have designed our circuit using logic works for 3 pipeline stages and it works fine. However, At this part of the project, we decided to go for more stages to ensure working close to the required frequency.



3 pipelined stages



4 pipelined stages

II. PART TWO

THEORITICAL CALCULATIONS

Cl=2000fF
Fmax=800MHz → Tclk=1/Fmax = 1250 ps
By referring to 0.5U Technology specification file we have:
Cox = 2.53 ff/µm2
Vdd = 5 V (Specified in the project)
Vtn =0.65V
Lmin = 0.6µm
Idsat = 400 µA/µm

We used a program attached to this document to do our calculations fast. The program implements the following formulas:

Wneff = (Vdd-Vtn) / (2 * Idsat * [Td / Cl]) .....(1.1)
Wpeff = 2.5 Wneff .....(1.2)
Cin= Cox * L * ∑ (Wnx+Wpx) <x runs from 1 to number of fan out> .....(1.3)

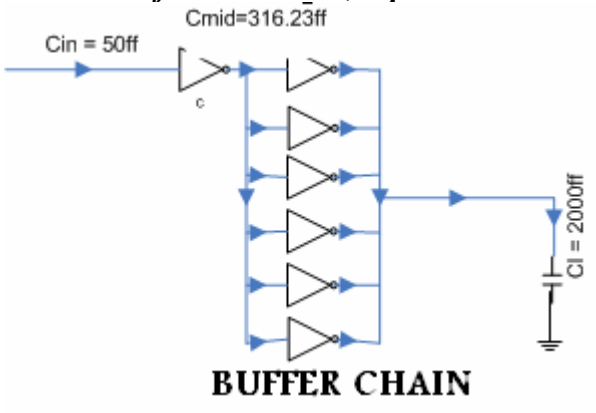
1 Buffer chain design

we want to reduce the load capacitance of the multiplier. We chose to make it Cl = 50fF.
Using buffer chain with n=2 inverters, we will have the following sizes for the inverter drived by the outputs:
Cin=Cox * Lmin * (Wn+Wp) → Wn+Wp = 50/(2.53 *0.6) = 33Um
But Wp=2.5Wn → Wn=9.4Um, Wp=23.5Um

The following inverter in the chain will have the following input capacitance:
Cin/Cmid =Cmid/ Cl → Cmid**2 = 50*2000 → Cmid = 316 ff
Since this input capacitance is approximately 6 times the one for the previous inverter, the inverter size will be 6 times larger than the previous one. To maintain smallest possible size, we make the first inverter drives **6** other inverters in parallel of similar sizes to the first one as indicated in fig1.

Now, we know that Td=Cl Requ ; where Req = (Vdd-Vtn) / 2(Idsat*Wneff)
By substituting for the first inverter in the chain, we find that its delay = 190ps
By substituting for the second stage in the chain, we find that its delay = 1208ps
Td total = 190+1208 = 1398ps
However, the clk period is 1250ps. So there is a 12% increase in the delay which must be considered in the SPICE simulation.

Concluding with inverter size of : Wn=9.4Um, Wp=23.5Um

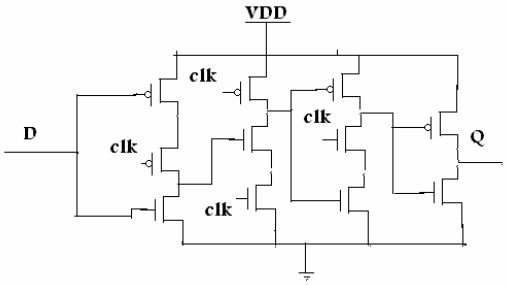


2 The multiplier circuit design

In order to calculate the required sizing, we consider the longest path in each stage. Upon that, the longest path consists of a flip flop and 3 full adders as shown in fig (1).
Now; Based on **Tclk >= Td (logic) + Tsetup + Tck-Q**:
We left a margin of 250ps of the clock and remained with 1000ps. We assigned 25% of the clock period to the setup time and clock to Q time of the flip flop. So,
Tck-Q+Tsetup =250ps and Td of the logic (3 full adders=750ps).
Now we start with the flip flop sizing calculations.

2.A D Flip Flop design

We used the True Single-Phase Clocked FF (TSPC FF) which is implemented as:



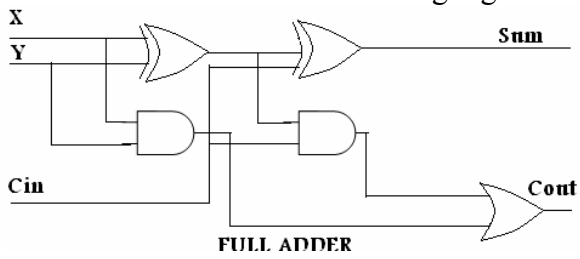
This implementation has 4 transistor levels. We assumed the following:

1- every level have equal delay time → Td = 250/4 =62.5ps ~ 60ps
2- the buffers in the implementation would behave as an inverter when they are enabled.

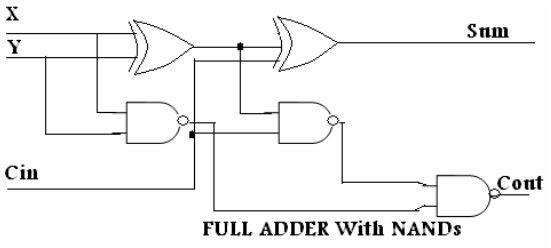
Now, this FF drives a load = 50ff . using equation 1.1 with Cl=50ff and Td=60ps, the size of the driving inverter (at Q) is : Wn =4um, Wp=12um.
Also the load to the next buffer = 28 ff
Since the size is less than the inverter size we achieved in the buffer chain, and since we want to have fixed sizes for every gate with the worst (largest) size as the best approximation, we made this inverter have the same buffer chain inverter size.
We notice that the load decreased as we advance backward that makes the sizes to even decrease more. Therefore, we went for the inverter size for all levels .
Concluding with inverter size of : Wn=9.4Um, Wp=23.5U
Buffers: Wn=9.4Um, Wp=23.5Um

2.B Full Adder design:

We considered 3 full adders (in the longest stage of the longest path) with Tdtotal=750ps (leaving some margins)
We assumed that every full adder will have equal delay = 750/3 =250 ps
Now we consider on full adder which has the following logic:

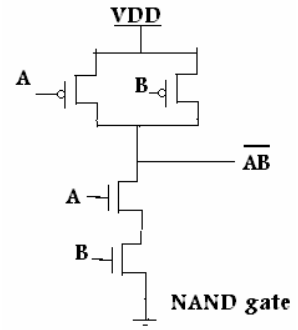


However we can do LOGIC TRANSFORMATION to achieve COMS implementation as follows:



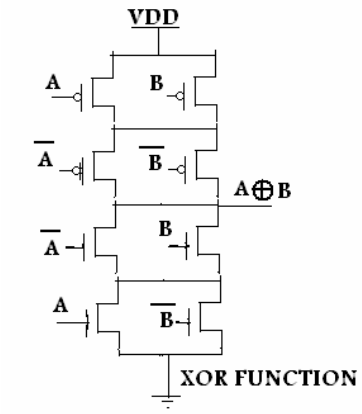
four levels can be seen in the full adder: NAND-NAND-XOR-INVERTER (this is the one which inputs to the XOR the complemented inputs). Assigning for every level equal delay results in ~ 60ps per level.

2.B.1 NAND gate design:



The NAND at Carry out sees a load of 50ff from the buffer in the D-FF. therefore, using equations 1.1, 1.2 and 1.3 with Cl=50ff and Td=60ps gives:
Wn=9µm Wp=11.33µm Cin =30.86ff. Now, since the following NANDs drives this 30.86ff<50ff → they will have smaller sizes. →
Concluding with NAND size of : Wn=9Um, Wp=11.33U

2.B.2 XOR design:



The XOR in the middle has a fan out=2 so it has more Load capacitance which implicate that it is larger than the one at the sum output. Its load comes from an inverter(the one after the XOR at the sum) =50ff and from the NAND with 30.86ff..
This results in a size: Wn =14µm Wp=37µm Cin=50ff
NOTE: Here we assume that the inverters are of the same size of the one calculated previously→ Td of INVwith (cload=50ff=cin(of XOR)) ~ 30ps
Now, since we assign 60ps initially for the inverter, we can give 60-30=30ps for the XOR to have a delay = 30+60=90ps.
→ recalculating the size produces: Wn=10µm Wp=25µm.
Concluding with NAND size of : Wn=10µm , Wp=25µm

This is a summary of the theoretical results achieved before moving to SPICE.

	Wn	***Wp***
INV	9.4µm	23.5µm
NAND	9µm	11.33µm
XOR	10µm	25µm
BUFFER(in D-FF)	9.4µm	23.5µm

NOTE:
We used these gates in the remaining logic like AND in the partial product generator. At this point we were able to design the remaining blocks using only these 4 basic units.

III. PART THREE

CIRCUIT DESIGN AND SIMULATION USING SPICE

1-Circuit design:

The SPICE file including all subcircuits used to design the multiplier are available in appendix #1.
After we did different tests and simulations(Some examples are provided in the next section) we came up with the following sizes and measures:

Operating frequency ~ 833 MHz

	Wn	***Wp***
INV	9µm	23µm
NAND	9µm	11µm
XOR	10µm	27µm
BUFFER(in D-FF)	9µm	23µm

These results are very close to our calculations. Which may be as a result of many assumption of underestimates and going to cases worse than the actual cases that results in such acceptable sizes.

***Total number of transistors:**

Basic units:
INV:2
DFF: 11
XOR+INV: 10
NAND:4
AND: 6
ONE BUFFER CHAIN:14
FULL ADDER: 2XOR+3NAND=2x10+3x4=32

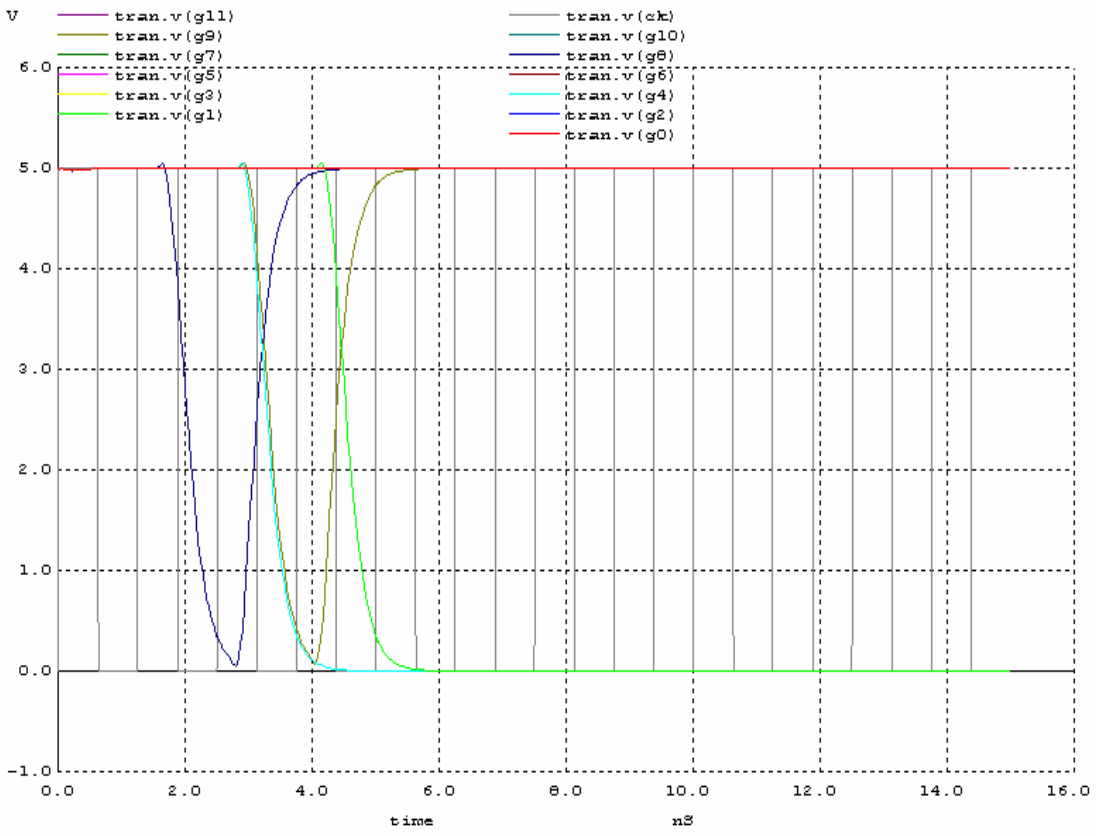
Building Blocks:
ADDERS: 30FULL ADDERS=30x32=960
REGISTERS: (12+12+6+14+6+14+2+12)DFF=78DFF=858
LOAD REGISTER: (12DFF+36NAND+1INV)=278
PPG: 36 AND= 216
12 BUFFER CHAINS:12x14=168

TOTAL=960+858+278+216+168=**2480 TRANSISTORS**

2.Circuit Simulation:

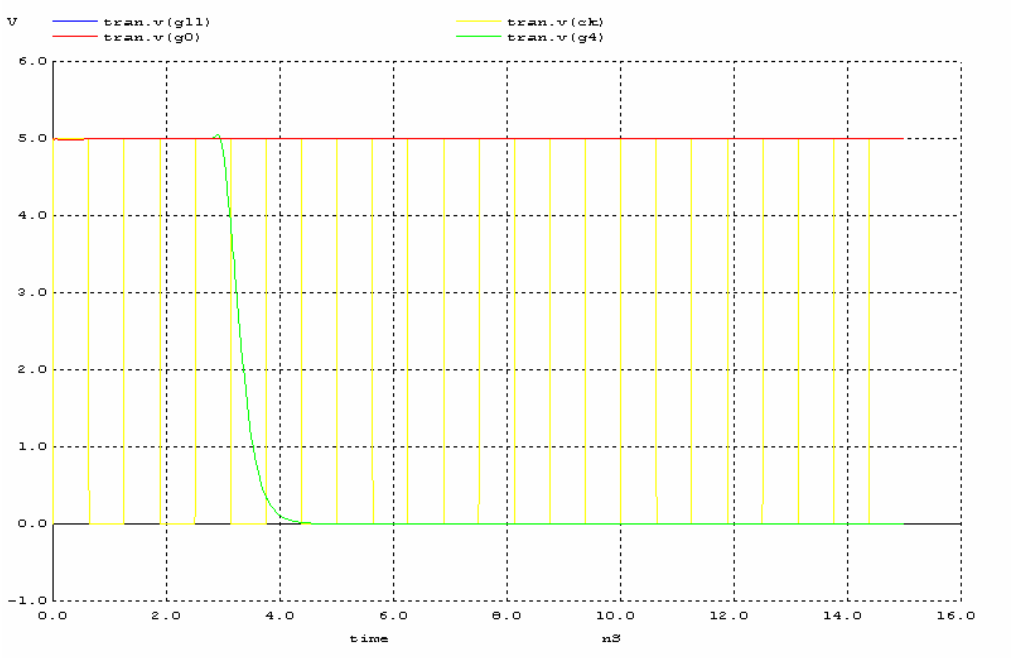
We have four pipeline stages + buffer delay . Each stage takes one **T =1.25ns.** So the result of multiplication for the first input has to appear after at most **5T= 6.25 ns**. When we simulated the circuit using transient analysis for some experimental inputs we got the following results:

Example 1: simulating: 111111*111111 = 111110000001
In the figure, g0 is the LSB and g11 is the MSB:



Analysis:
We notice that the outputs are totally stabilized before 6ns(after 4 T's) which means that it works at a frequency around 1/(6ns)/(4stages+buffer dealay) =833.33MHz

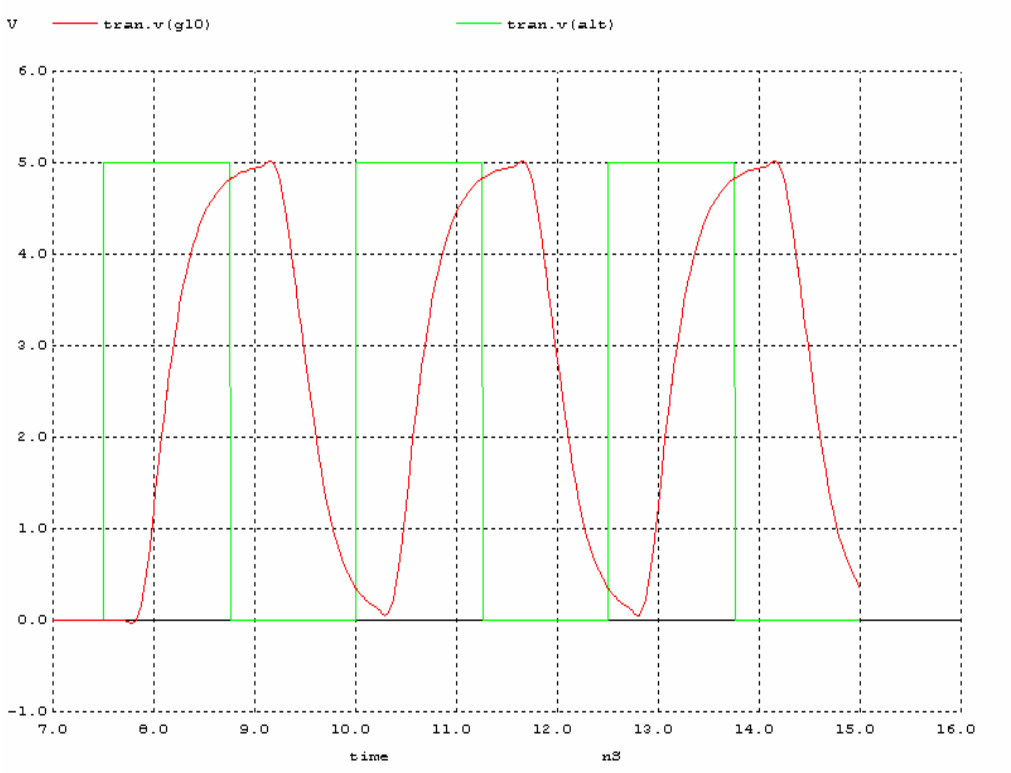
Example 2: simulating the bits g0,g4,g11 in the product:
111011*101011=100111101001
G0=1
G4=0
G11=1



Analysis:
After 5 clock cycles, the outputs are correctly stable at the logic levels. Since we expect the first output to appear after 5 clocks (4stages+buffer delay), we can approve that the multiplier works at f=800MHz.

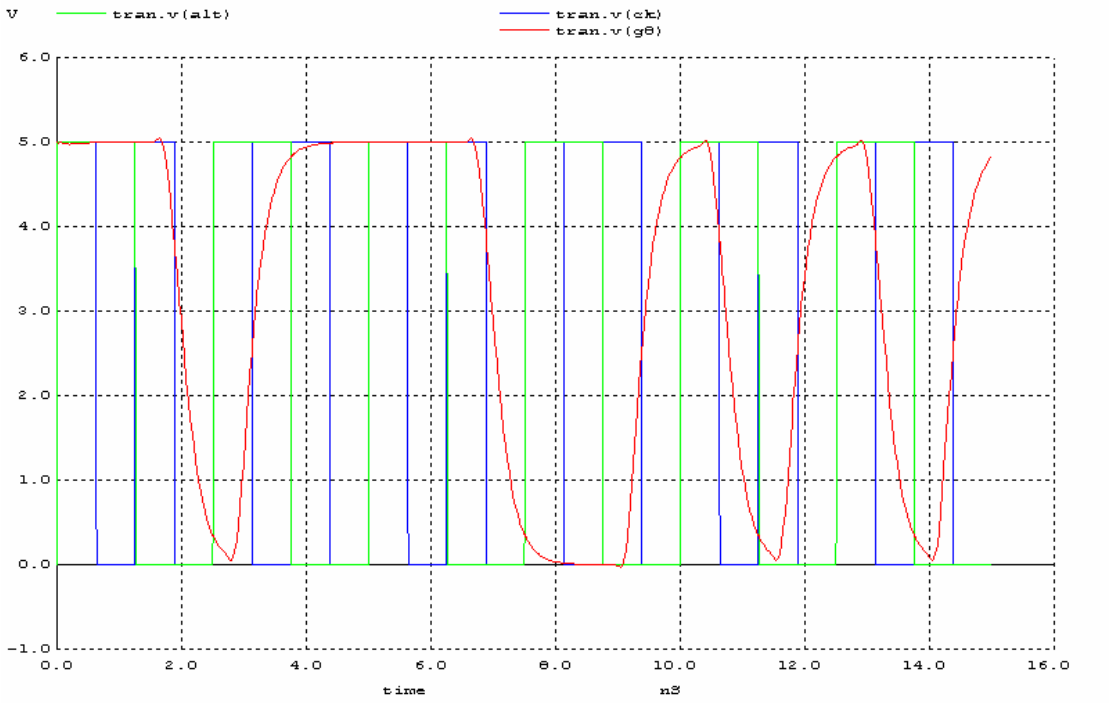
Example 3: Pipelining Test:
If the pipelining principle is designed correctly, then applying new inputs after every clock cycle will make the results to change correctly after **one clock cycle** starting from the first output which takes the longest period.

We tested the following inputs:
A=111000 * B=100001 = **1**1100 1 11000
A=011000 * B=100001 = **0**1100 0 11000
ALtering A5: Valt ALT 0 DC 0 pulse(0 5 0 0.005ns 0.005ns 1.249ns 2.5ns)
Altering bit 5 in A will cause bit 10 in the result to change every clock cycle between 0 and1. After simulation we got:



Analysis:
If we accept that Logic 1=(3V,5V) and Logic 0=(0V, 2V) as assumed in this report and as the circuit designed around that, we notice that when the input (Valt) = 1, the output (g10) goes high and when the input (Valt) = 0, the output (g10) goes low. The differenc in time between the change in the result, from min V(g10) as high=3V to Max V(g10) as Low=2V is as follows:
Consider the third input(alt) period, Vg10 = 3V at 12ns Vg10=2V at 13.2ns
→ period of change = 13.2- 12= 1.2ns→ f = 1/1.1ns ~ 833MHz

Example 4: Pipelining Test:
We applied the same idea in the previous example with the following data:
A=111 **1** 00 * B=111111 = 111 **0** 11000100
A=111 **0** 00 * B=111111 = 110 **1** 11001000
ALtering A2: Valt ALT 0 DC 0 pulse(0 5 0 0.005ns 0.005ns 1.249ns 2.5ns)
Altering bit 2 in A will cause bit 8 in the result to change every clock cycle between 0 and1. After simulation we got:



Analysis:
1-The output(g8) starts to appear correctly (0 at the first time) at t~7ns (after 5.5T). However we expect it to appear after 5T. this difference is due to the delay of the buffer chain that we first calculated(See page5). This buffer will contribute a delay of more than 1250ps (more than 1T) and the result shwed that.

2-doing the same analysis in example 3, we got that time difference between High g8 and Low g8 in consecutive alternating inputs is= 1.2→ f 833MHz

3. CONCLUSION:

1-SPICE had proved fast and reliable simulation results.

2-as we increase the pipeline stages we may achieve better results but the number of transistor used would also increase.

3-the load capacitance has a very important effect on determining the operating frequency. As we can decrease this load capacitance(using buffer chains), we can have more space of delay considerations to assign per each gate. This results in better sizing.

4-the use of buffer chain is chief to get small load capacitance to the circuit. As we wish to have small load, the buffer chain increases in size. This of course, leads to the problem of adding a delay before the output can be conducted. So, there has to be a sort of trade-off in dealing with buffer chains.

5-as the transistors increase in size, the performance (operating frequency in our case) increases too. However, due to manufacturing and power consumption considerations, we cannot go for very large sizes. Our results were, somehow, reasonable.

APPENDIX A
SPICE FILES

See next 14 pages numbered (1-14).

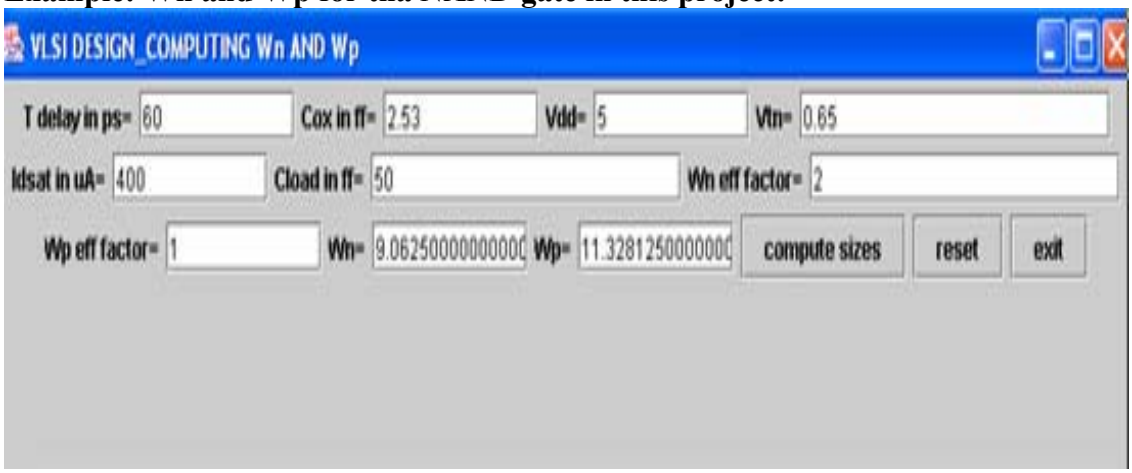
APPENDIX B
Simple Java GUI program for calculating Wn and Wp

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
public class VLSIPROG extends JFrame implements ActionListener{
        double a,b,c,d,e,f,g,h;
        private JPanel jp=new JPanel();
        private JLabel l1=new JLabel("T delay in ps=");
        private JTextField j1=new JTextField(10);

        private JLabel l2=new JLabel("Cox in ff=");
        private JTextField j2=new JTextField(10);

        private JLabel l3=new JLabel("Vdd=");
        private JTextField j3=new JTextField(10);

        private JLabel l4=new JLabel("Vtn=");
        private JTextField j4=new JTextField(20);

        private JLabel l5=new JLabel("Idsat in uA=");
        private JTextField j5=new JTextField(10);

        private JLabel l6=new JLabel("Cload in ff=");
        private JTextField j6=new JTextField(20);

        private JLabel l7=new JLabel("Wn eff factor=");
        private JTextField j7=new JTextField(20);

        private JLabel l8=new JLabel("Wp eff factor=");
        private JTextField j8=new JTextField(10);

        private JLabel l9=new JLabel("Wn=");
        private JTextField j9=new JTextField(10);

        private JLabel l10=new JLabel("Wp=");
        private JTextField j10=new JTextField(10);

        private JButton b1=new JButton("compute sizes");
        private JButton b2=new JButton("reset");
        private JButton b3=new JButton("exit");
        public VLSIPROG(){
                super("VLSI DESIGN_COMPUTING Wn AND Wp");
                setSize(1000,200);
                Container cp=getContentPane();
                cp.setLayout(new FlowLayout());

                cp.add(l1);
                cp.add(j1);
                cp.add(l2);
                cp.add(j2);
                cp.add(l3);
```

```
                cp.add(j3);
                cp.add(l4);
                cp.add(j4);
                cp.add(l5);
                cp.add(j5);
                cp.add(l6);
                cp.add(j6);
                cp.add(l7);
                cp.add(j7);
                cp.add(l8);
                cp.add(j8);
                cp.add(l9);
                cp.add(j9);
                cp.add(l10);
                cp.add(j10);
                cp.add(b1);
                cp.add(b2);
                cp.add(b3);
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
                show();

        }

        public void actionPerformed(ActionEvent ae){

                if(ae.getSource()==b1){
                        a=Double.parseDouble((j1.getText()).trim());
                        b=Double.parseDouble((j2.getText()).trim());
                        c=Double.parseDouble((j3.getText()).trim());
                        d=Double.parseDouble((j4.getText()).trim());
                        e=Double.parseDouble((j5.getText()).trim());
                        f=Double.parseDouble((j6.getText()).trim());
                        g=Double.parseDouble((j7.getText()).trim());
                        h=Double.parseDouble((j8.getText()).trim());
                        j9.setText(m1(a,b,c,d,e,f,g,h));
                        j10.setText(m2(a,b,c,d,e,f,g,h));
                }
                else if(ae.getSource()==b2){
                        j1.setText("");
                        j2.setText("");
                        j3.setText("");
                        j4.setText("");
                        j5.setText("");
                        j6.setText("");
                        j7.setText("");
                        j8.setText("");
                        j9.setText("");
                        j10.setText("");
                }
                else System.exit(0);
        }
        public String m1(double a,double b,double c,double d,double e,double f,double
g,double h){
```

```
        double wn=(g*(c-d)*f*(Math.pow(10,-15)))/(2*e*(Math.pow(10,-
6))*a*(Math.pow(10,-12)));
        return Double.toString(wn);
        }
        public String m2(double a,double b,double c,double d,double e,double f,double
g,double h){
        double wn=(g*(c-d)*f*(Math.pow(10,-15)))/(2*e*(Math.pow(10,-
6))*a*(Math.pow(10,-12)));
        double wp=2.5*h*wn/g;
        return Double.toString(wp);
        }


        public static void main(String args[]){
                new VLSIPROG();
                }
}
```

Example: Wn and Wp for tha NAND gate in this project:



REFERENCES

1. Leblebici, Yusuf and Sung-Mo Kang. CMOS Digital Integrated Circuits Analysis and Design. McGRAW-HILL: 1999

2. M. Morris Mano, Charles R. Kime. Logic and Computer Design Fundamentals. Printice Hall:2001

3. Dr. Elrabaa lecture notes on VLSI design course (COE 360).