**EE5423-Computer Arithmetic**

**Name: Vinay Chittam**

**Student ID: Fet110**

**Project #2**

**Aim**: The goal of the project is to compare the performance of an unsigned add and shift multiplier with one using a shift-over technique.

**Softwares used:** C language and Excell.

**Procedure**: For an unsigned shift and add multiplier, we are only concerned about positive numbers. The multiplication is nothing but simple add and shift process. We use an RCA(ripple carry adder) to do the addition process. The delay of a RCA is given by the formula “n(2d+2d+2nd)”, where n= number of bits. The first 2d is for multiplexing. The multiplexing works the following way: First we scan the LSB in multiplier and decide what action is to be performed. This action of sending either “0”, or “A” to perform addition with a temporary result is called multiplexing, which takes a delay of 2d. The next 2d is for shifting the bit. This is performed after obtaining the sum, which is performed by the RCA. The shift takes 2d delay. The other 2nd is the worst case delay for RCA. We are not interested in calculating the actual delay of the adder. By using the above formula we can calculate all the delay for different values of n, where n varies from 2 to 32 bits. This is can be shown graphically by plotting it in excel. The graph is shown below. The Y- axis is plotted in logarithmic scale.

**Code**: #include<stdio.h>

#include<conio.h>

void main(){

int n,delay;

clrscr();

for(n=2;n<=32;n++){

if(n==18)

getch();

delay=(n\*(2+2+(2\*n)));

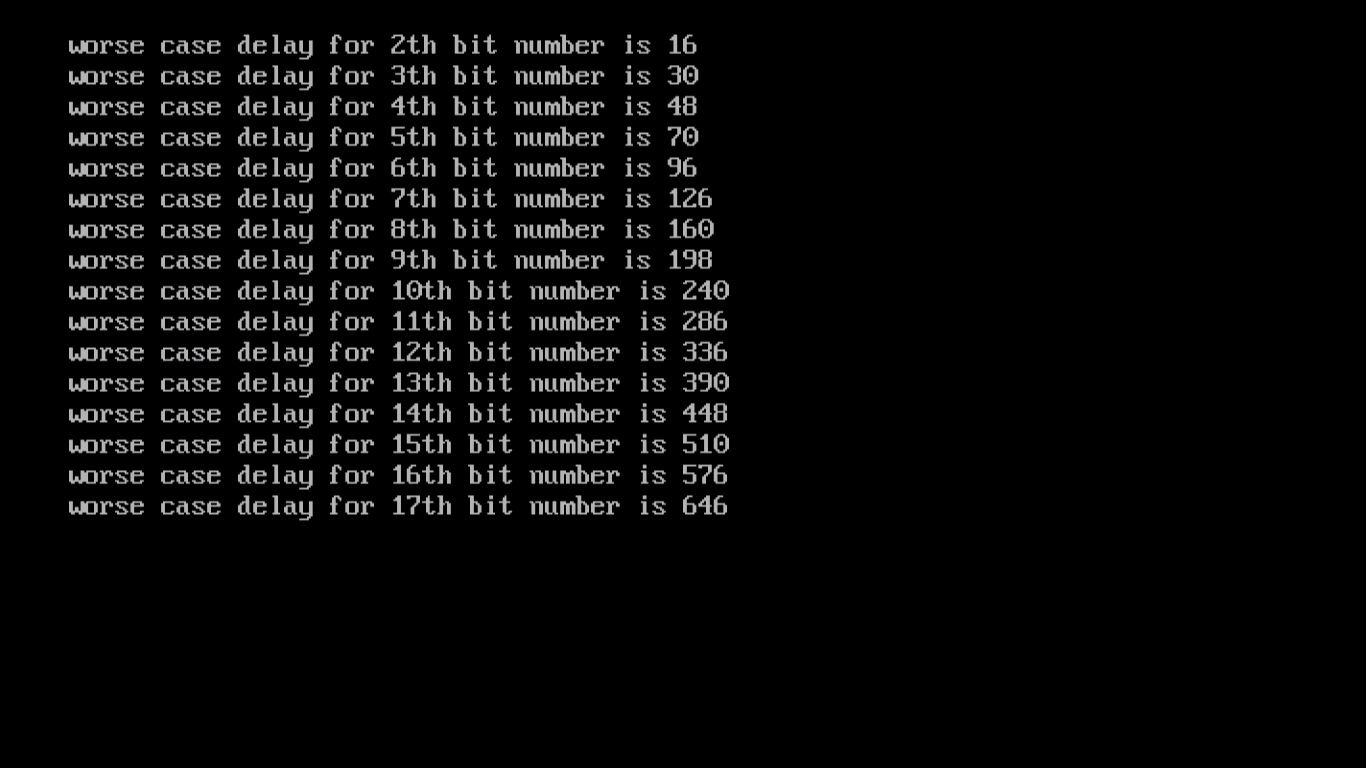
printf("\nworse case delay for %dth bit number is %d",n,delay);

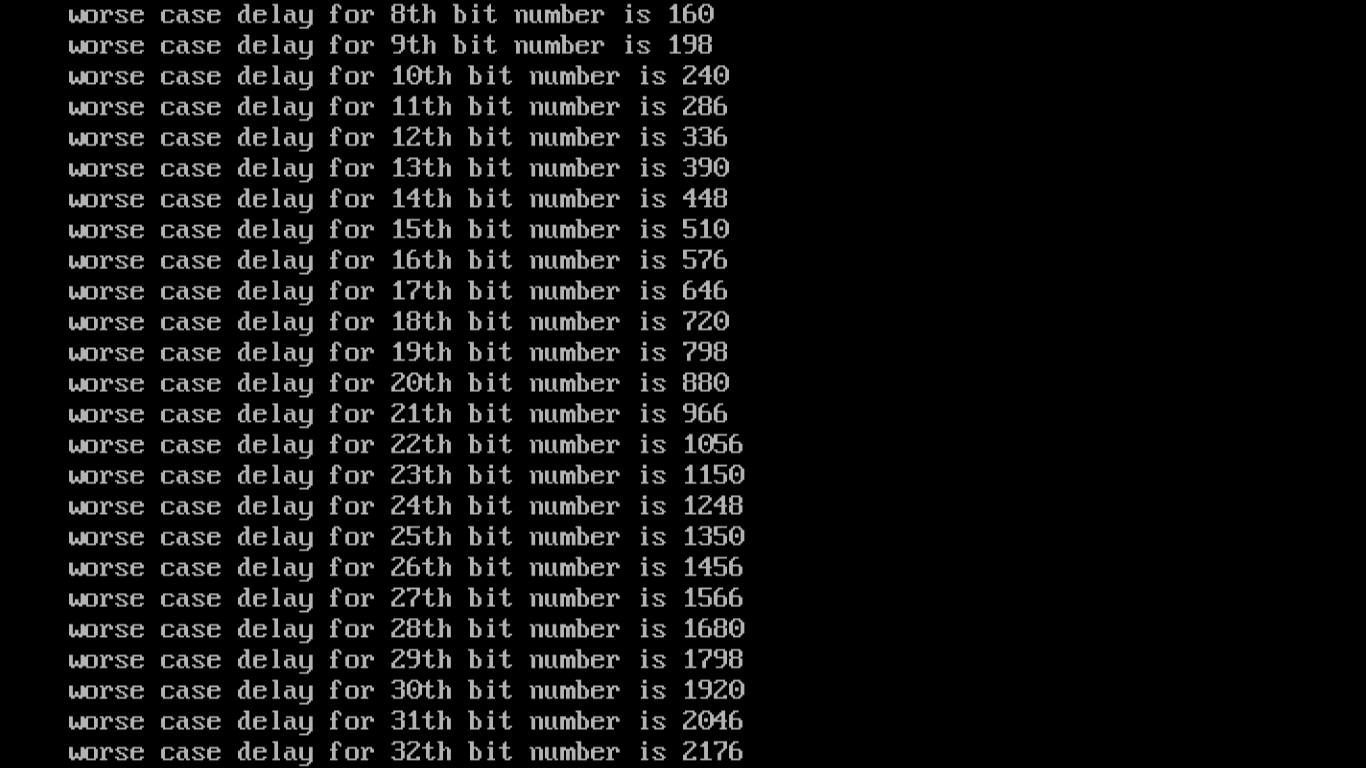
}

getch();

}

Output:





**Simulation of shift and add asynchronous multiplier with shift over 0’s and 1’s:** In this process we save the number of hardware used as well as delay to calculate the multiplication. This is done by shifting of 1’s or 0’s when they are occurring continuously. Even this process is unsigned. So we need to just deal with positive numbers. We use CCA(carry completion adder) to perform the addition process. We perform the multiplexing by adding either +A or –A depending of the present and previous bit in the multiplier B. The truth table to decide the action is given as follows.

Present bit Past bit Action

0 0 0

0 1 A

1 0 -A

1 1 0

So for this to happen, I created a function to preprocess –A. We use 2’s complement adder subtractor to perform addion or subtraction and since it is addition of subtraction, we always pad a 0 if A or B has a bit “1 ” in the MSB while getting generated using random function. This is done to always make sure that the final result is positive unsigned number. And since its 2’s complement process, we ignore the carry after performing the addition. We use the CCA process to calculate the delay for all the carry bits to get finalzed. The overall delay for the multiplication process in this technique involves the following delay 1. Multiplexer delay, 2. Carry completion delay, 3. Shift Delay , and 4. Triggering delay.

The multiplexer delay is again 2d delay and occurs when ever there is an action performed. i.e. when ever we the multiplexer select + or –A depending of the LSB of the multiplier B, a 2d is added for multiplexing.

Some amount of delay occurs for CCA addition, which depends on the operands on which CCA performed, and this delay is calculated by the CCA process used in project 1 earlier. It takes care of calculating the delay of addition process and generates the intermediate results.

The intermediate result is shifted, and this shift takes another 2d delay. It is called as shift delay.

Finally we have another 2d delay to trigger the next cycle to begin.

Thus the over delay for the multiplication process is as follows,

1 2d delay for mux,

2. Some amount of delay for CCA addition

3. 2d delay for shifting the intermediate result

4. 2d delay for triggering the next cycle.

All these delays have to be calculated for each bit of multiplier.

If the bits of multiplier are continuous i.e. if we have a continuous strings of 0’s or 1’s we just shift the bits which takes 2d delay and we don’t perform multiplexing of CCA process to those bits which are repeated. This is why the process is called as the shift over technique.

We find the delay for number of bits varying from n=2 to n=32. And for each n, we calculate the delay 10,000 times, using random function to generate the values of A and B. For each n, we then calculate average delay. After calculating this delay, we compare the results, with that of the results generated from technique 1, which uses RCA process for addition.

**Code :**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<alloc.h>

int ccad,muxd,shiftd,trigd,\*sum,\*a,\*b,\*a1,\*b1,\*cc1,\*cc0,\*c0,\*c1,\*t0,\*t1,\*res,resi,i,j,k,n,s,m,l,t,td,flag,c0f,c0l,c1f,c1l;

long ttd;

float avgd;

void adder();

void setcc0cc1();

void set();

void bits(int \*cc0,int \*cc1,int \*t0,int \*t1,int \*a,int \*b);

int\* comp2();

void freemem(){

free(cc0);free(cc1);free(c0);free(c1);free(t0);free(t1);free(a);free(b);free(sum);free(res);

}

void rtshift(int \*p,int n){

int i;

shiftd+=2;

trigd+=2;

res[resi]=p[n];

resi--;

for(i=n;i>0;i--){

p[i]=p[i-1];

}

}

void mul(){

int i=n,j,k;

int pre=0,curr=i;

for(i=n,curr=i,pre=0;i>=0;i--,curr--){

if(b[curr]==0 && pre==0){

res[i]=c1[i];

pre=b[curr];

rtshift(c1,n);

continue;

}

else if(b[curr]==1 && pre==1){

res[i]=c1[i];

pre=b[curr];

rtshift(c1,n);

continue;

}

else if(b[curr]==0 && pre==1){

for(j=0;j<=n;j++){

c0[j]=a[j];

}

}

else if(b[curr]==1 && pre==0){

c0=comp2(a,n+1);

}

muxd+=2;

setcc0cc1();

do{

bits(cc0,cc1,t0,t1,c1,c0);

flag=0;

for(k=0;k<=n+1;k++){

if(cc0[k]==cc1[k]){

flag=1;

set();

}

}

if(flag==0){ /\*

t++;

td+=2;

ttd+=td; \*/

break;

}

else{ /\*

t++;

td+=2; \*/

}

ccad+=2;

}

while(flag==1);

adder();

for(k=0;k<=n;k++){

c1[k]=sum[k];

}

rtshift(c1,n);

pre=b[curr];

}

}

void adder(){

ccad+=2;

for(i=n;i>=0;i--)

sum[i]=c1[i]^c0[i]^cc1[i+1];

//printf("\n");

}

void set(){

for(j=0;j<=n+1;j++){

t1[j]=cc1[j];

t0[j]=cc0[j];

}

}

void bits(int \*c0,int \*c1,int \*t0,int \*t1,int \*a,int \*b){

for(i=n;i>=0;i--){

s=a[i]+b[i];

if(s==0){

c0[i]=1;

c1[i]=0;

}

else if(s>1){

c0[i]=0;

c1[i]=1;

}

else if(s==1){

if(t0[i+1]==t1[i+1]){

c0[i]=0;

c1[i]=0;

}

else{

m=s+t1[i+1];

if(m>1){

c0[i]=0;

c1[i]=1;

}

else if(m<=1){

c0[i]=1;

c1[i]=0;

}

}

}

}

}

void setcc0cc1(){

for(i=0;i<=(n+1);i++){

t1[i]=0;

t0[i]=0;

cc1[i]=0;

cc0[i]=0;

}

//printf("\n %d...%d...%d......",t1[n+1],t0[n+1],n);

t1[n+1]=0;

t0[n+1]=1;

cc1[n+1]=0;

cc0[n+1]=1;

ccad+=2;

}

int\* comp2(int \*p,int last){

int i,carry=0,n=last;

int \*c0;

//p=(int \*)calloc(n,sizeof(int));

c0=(int \*)calloc(n,sizeof(int));

for(i=last-1;i>=0;i--){

if(p[i]==0){

c0[i]=1;

}

else if(p[i]==1){

c0[i]=0;

}

}

for(i=last-1;i>=0;i--){

if(i==last-1){

if(c0[i]==0){

c0[i]=1;

}

else if(c0[i]==1){

c0[i]=0;

carry=1;

}

}

else{

if(c0[i]==0 && carry==1){

c0[i]=1;

carry=0;

}

else if(c0[i]==1 && carry==1){

c0[i]=0;

carry=1;

}

}

}

return c0;

}

void main(){

//int pre=0,curr=i;

int x,y,z,loop=10;

unsigned long totaldelay;

float avgdelay;

//printf("execution begins:");

clrscr();

z=3;

t=0,ttd=0;

for(z=2;z<=32;z++){

a=(int \*)malloc((z+1)\*sizeof(int));

b=(int \*)malloc((z+1)\*sizeof(int));

c1=(int \*)malloc((z+1)\*sizeof(int));

c0=(int \*)malloc((z+1)\*sizeof(int));

cc1=(int \*)malloc((z+2)\*sizeof(int));

cc0=(int \*)malloc((z+2)\*sizeof(int));

t1=(int \*)malloc((z+2)\*sizeof(int));

t0=(int \*)malloc((z+2)\*sizeof(int));

sum=(int \*)malloc((z+1)\*sizeof(int));

res=(int \*)malloc((z+z+2)\*sizeof(int));

resi=z+z+1;

ttd=0;

totaldelay=0;

n=z;

//ccad=0,muxd=0,shiftd=0,trigd=0;

for(x=0;x<loop;x++){

ttd=0,ccad=0,muxd=0,shiftd=0,trigd=0;

resi=n+n;

td=2;

//a[0]=0;a[1]=1;a[2]=0;a[3]=0;a[4]=1;a[5]=0;a[6]=1;a[7]=1;a[8]=0;a[9]=1;a[10]=1;a[11]=1;//a[12]=1;//a[13]=0;a[14]=1;a[15]=1;a[16]=0;a[17]=1;a[18]=1;a[19]=0;a[20]=0;a[21]=1;a[22]=0;a[23]=1;

//b[0]=0;b[1]=1;b[2]=1;b[3]=0;b[4]=1;b[5]=1;b[6]=0;b[7]=0;b[8]=1;b[9]=0;b[10]=0;b[11]=1;//b[12]=1;//b[13]=1;b[14]=0;b[15]=0;b[16]=1;b[17]=0;b[18]=0;b[19]=0;b[20]=1;b[21]=0;b[22]=1;b[23]=1;

for(y=0;y<n+1;y++){

a[y]=0;

b[y]=0;

c0[y]=0;

c1[y]=0;

sum[y]=0;

}

for(y=0;y<n+2;y++){

t0[y]=0;

t1[y]=0;

cc0[y]=0;

cc1[y]=0;

//sum[y]=0;

}

t1[z+1]=0;

t0[z+1]=1;

cc1[z+1]=0;

cc0[z+1]=1;

for(y=0;y<=n+n+1;y++)

res[y]=0;

for(y=1;y<n+1;y++){

a[y]=((rand())%2);

b[y]=((rand())%2);

}

a[0]=0;

b[0]=0;

//printf("\n");

mul();

resi++;

for(i=n;i>=0;i--){

res[resi]=sum[i];

resi--;

}

ttd=ccad+shiftd+trigd+muxd;

totaldelay+=ttd;

//printf("\nfinal delay for %dth bit till %dloop=%ld",n,x,totaldelay);

}

//printf("\n");

//printf("\ntotal delay for %dth bit number is d",n);

freemem();

//totaldelay+=ttd;

avgdelay=(float)totaldelay/loop;

//printf("\n");

printf("\ntotal delay for %dth bit number = %lud and avg delay = %f",z,totaldelay,avgdelay);

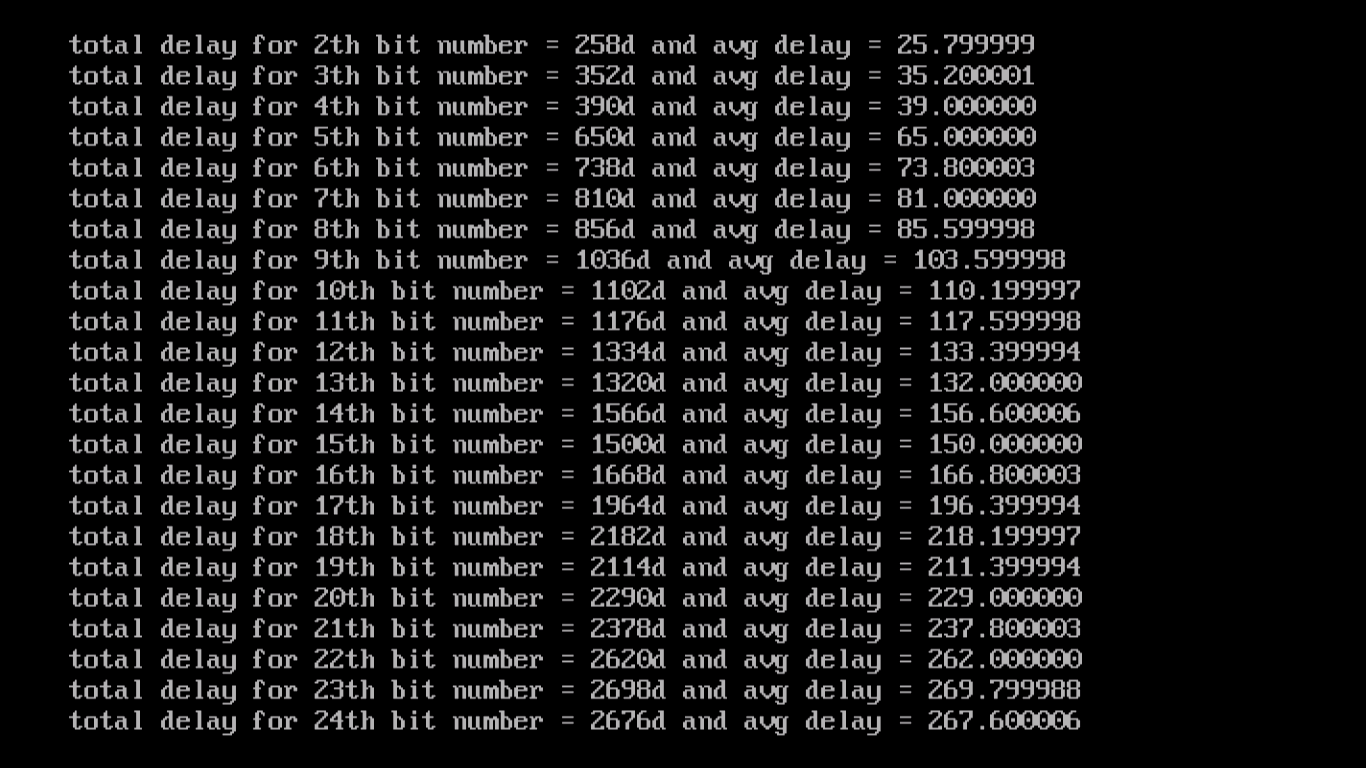
getch();

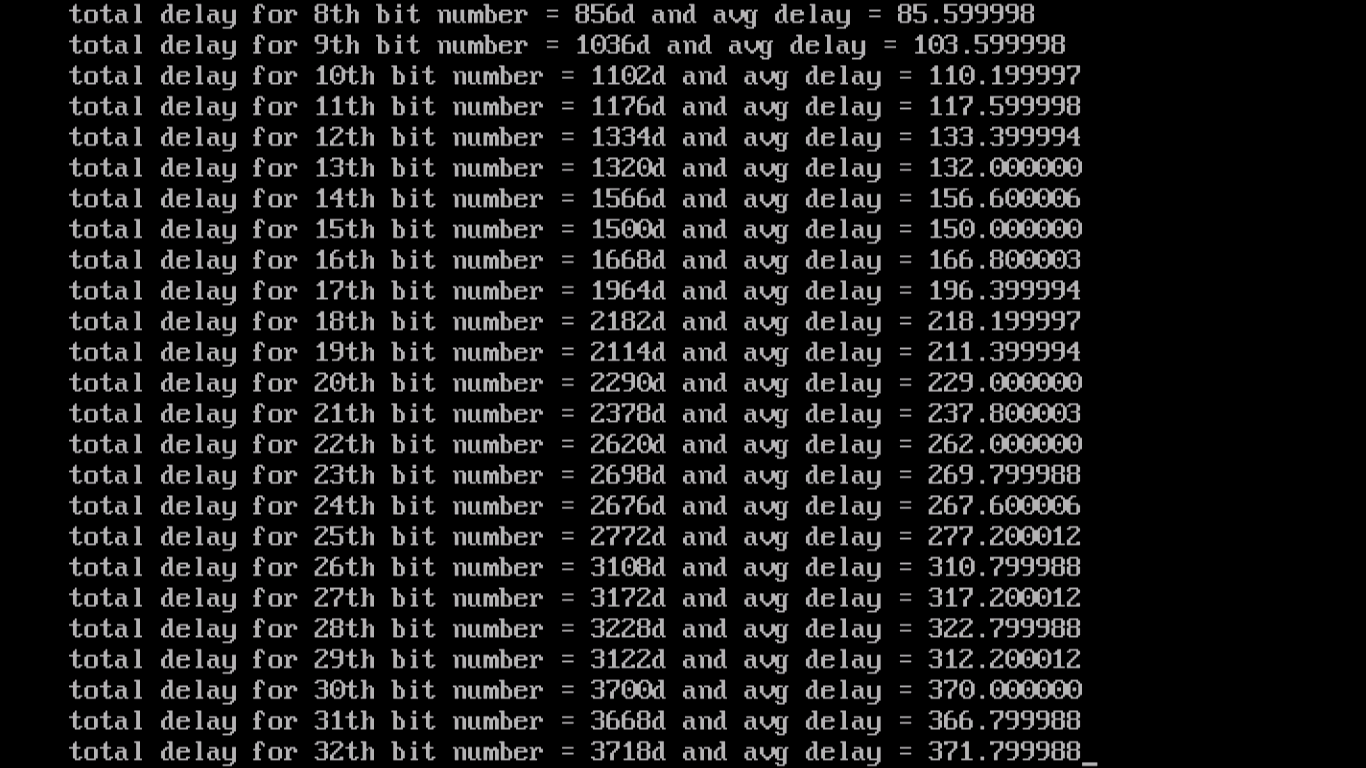
}

getch();

}

The output of the simulation is shown below:





Note: I have run the simulation only 10 times, as it was taking a lot of time to compute all the loops.

**Comparison of first and second technique:**

Note: While comparing the performance of both the techniques, I have changed the graph of first technique. i.e. I changed the graph of Y axis from log scale to normal scale.

We can switch the scale from log to normal or vice versa by just right clicking any value of y axis and then select “LOGs scale”. For convenience I have chosen a normal scale. As mentioned in the question for the first technique, I have used the log scale, a few pages above.

Observation: It can be observed that, the RCA in multiplication process, takes a lot of delay, where as the shift over techniques saves a lot of time in computation. It can be clearly seen from the graph that for n=32 the shift over takes very less delay, where as the first technique takes relatively longer time. It takes less delay because of the shifting over process, which even reduces the number of multiplexers used. The number of additions also are skipped when the shift over occurs.

**Performance of Shift and add asynchronous multiplier with shift over 0’s and 1’s for a 12 bit number:**

The given number is x4b7 and x6c9, which is represented in binary format as follows:

A=010010110111 and B=011011001001. We perform the process of multiplication as explained above on these numbers. The delay is shown in steps of 2d.

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<alloc.h>

int ccad,muxd,shiftd,trigd,\*sum,\*a,\*b,\*a1,\*b1,\*cc1,\*cc0,\*c0,\*c1,\*t0,\*t1,\*res,resi,i,j,k,n,s,m,l,t,td,flag,c0f,c0l,c1f,c1l;

long ttd;

float avgd;

void adder();

void setcc0cc1();

void set();

void bits(int \*cc0,int \*cc1,int \*t0,int \*t1,int \*a,int \*b);

int\* comp2();

void rtshift(int \*p,int n){

int i;

shiftd+=2;

trigd+=2;

res[resi]=p[n];

resi--;

for(i=n;i>0;i--){

p[i]=p[i-1];

}

}

void mul(){

int i=n,j,k;

int pre=0,curr=i;

for(i=n,curr=i,pre=0;i>=0;i--,curr--){

if(b[curr]==0 && pre==0){

res[i]=c1[i];

pre=b[curr];

rtshift(c1,n);

continue;

}

else if(b[curr]==1 && pre==1){

res[i]=c1[i];

pre=b[curr];

rtshift(c1,n);

continue;

}

else if(b[curr]==0 && pre==1){

for(j=0;j<=n;j++){

c0[j]=a[j];

}

}

else if(b[curr]==1 && pre==0){

c0=comp2(a,n+1);

}

printf("\nc1=");

for(k=0;k<n+1;k++)

printf("%d ",c1[k]);

printf("\tshift delay=2d,trigging delay=2d");

printf("\nc0=");

for(k=0;k<n+1;k++)

printf("%d ",c0[k]);

muxd+=2;

printf("\tmultiplex delay=2d");

setcc0cc1();

do{

bits(cc0,cc1,t0,t1,c1,c0);

flag=0;

for(k=0;k<=n;k++){

if(cc0[k]==cc1[k]){

flag=1;

set();

}

}

if(flag==0){

t++;

td+=2;

ttd+=td;

break;

}

else{

t++;

td+=2;

}

printf("\n..cc1=");

for(k=0;k<=n+1;k++)

printf("%d ",cc1[k]);

printf("\n..cc0=");

for(k=0;k<=n+1;k++)

printf("%d ",cc0[k]);

ccad+=2;

printf("\tcarry adder delay=2d");

//getch();

}

while(flag==1);

adder();

for(k=0;k<=n;k++){

c1[k]=sum[k];

}

rtshift(c1,n);

pre=b[curr];

}

}

void adder(){

printf("\nlast cc1=");

for(k=0;k<=n+1;k++)

printf("%d ",cc1[k]);

printf("\nlast cc0=");

for(k=0;k<=n+1;k++)

printf("%d ",cc0[k]);

ccad+=2;

printf("\tcarry adder delay=2d");

getch();

for(i=n;i>=0;i--)

sum[i]=c1[i]^c0[i]^cc1**[i+1];**

printf("\n");

}

void set(){

for(j=0;flag==1 && j<=n;j++){

t1[j]=cc1[j];

t0[j]=cc0[j];

}

}

void bits(int \*c0,int \*c1,int \*t0,int \*t1,int \*a,int \*b){

for(i=n;i>=0;i--){

s=a[i]+b[i];

if(s==0){

c0[i]=1;

c1[i]=0;

}

else if(s>1){

c0[i]=0;

c1[i]=1;

}

else if(s==1){

if(t0[i+1]==t1[i+1]){

c0[i]=0;

c1[i]=0;

}

else{

m=s+t1[i+1];

if(m>1){

c0[i]=0;

c1[i]=1;

}

else if(m<=1){

c0[i]=1;

c1[i]=0;

}

}

}

}

}

void setcc0cc1(){

for(i=0;i<(n+1);i++){

t1[i]=0;

t0[i]=0;

cc1[i]=0;

cc0[i]=0;

}

t1[n+1]=0;

t0[n+1]=1;

cc1[n+1]=0;

cc0[n+1]=1;

printf("\nfirst cc1=");

for(k=0;k<=n+1;k++)

printf("%d ",cc1[k]);

printf("\nfirst cc0=");

for(k=0;k<=n+1;k++)

printf("%d ",cc0[k]);

ccad+=2;

printf(" 2d for AND gate (bits finilized)");

}

int\* comp2(int \*p,int last){

int i,carry=0,n=last;

int \*c0;

//p=(int \*)calloc(n,sizeof(int));

c0=(int \*)calloc(n,sizeof(int));

for(i=last-1;i>=0;i--){

if(p[i]==0){

c0[i]=1;

}

else if(p[i]==1){

c0[i]=0;

}

}

for(i=last-1;i>=0;i--){

if(i==last-1){

if(c0[i]==0){

c0[i]=1;

}

else if(c0[i]==1){

c0[i]=0;

carry=1;

}

}

else{

if(c0[i]==0 && carry==1){

c0[i]=1;

carry=0;

}

else if(c0[i]==1 && carry==1){

c0[i]=0;

carry=1;

}

}

}

return c0;

}

void main(){

//int pre=0,curr=i;

clrscr();

n=11,t=0;

a=(int \*)calloc(n+1,sizeof(int));

b=(int \*)calloc(n+1,sizeof(int));

c1=(int \*)calloc((n+1),sizeof(int));

c0=(int \*)calloc((n+1),sizeof(int));

cc1=(int \*)calloc((n+2),sizeof(int));

cc0=(int \*)calloc((n+2),sizeof(int));

t1=(int \*)calloc((n+2),sizeof(int));

t0=(int \*)calloc((n+2),sizeof(int));

sum=(int \*)calloc((n+1),sizeof(int));

res=(int \*)calloc((n+n+2),sizeof(int));

resi=n+n+1;

ttd=0;

ccad=0,muxd=0,shiftd=0,trigd=0;

a[0]=0;a[1]=1;a[2]=0;a[3]=0;a[4]=1;a[5]=0;a[6]=1;a[7]=1;a[8]=0;a[9]=1;a[10]=1;a[11]=1;//a[12]=1;//a[13]=0;a[14]=1;a[15]=1;a[16]=0;a[17]=1;a[18]=1;a[19]=0;a[20]=0;a[21]=1;a[22]=0;a[23]=1;

b[0]=0;b[1]=1;b[2]=1;b[3]=0;b[4]=1;b[5]=1;b[6]=0;b[7]=0;b[8]=1;b[9]=0;b[10]=0;b[11]=1;//b[12]=1;//b[13]=1;b[14]=0;b[15]=0;b[16]=1;b[17]=0;b[18]=0;b[19]=0;b[20]=1;b[21]=0;b[22]=1;b[23]=1;

for(i=0;i<(n+1);i++){

c1[i]=0;

}

for(i=0;i<(n+n+1);i++){

res[i]=0;

}

printf("\na =");

for(k=0;k<(n+1);k++){

printf("%d ",a[k]);

}

printf("\nb =");

for(k=0;k<(n+1);k++){

printf("%d ",b[k]);

}

printf("\n");

mul();

avgd=(float)ttd/1000;

resi++;

for(i=n;i>=0;i--){

res[resi]=sum[i];

resi--;

}

//printf("\nat iteration %d ",t);

printf("\nfinal sum is :");

for(i=0;i<=(n+n+1);i++){

printf("%d ",res[i]);

}

ttd=ccad+shiftd+trigd+muxd;

printf("\n");

printf("\ntotal delay for %dth bit number is %dd",n,ttd);

getch();

}

