ΕΡΓΑΣΤΗΡΙΟ 3

04/04/2021

ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ – ΔΗΜΙΟΥΡΓΙΑ DATASET ΓΙΑ ΤΟΝ ΑΠΛΟ ΠΡΟΣΟΜΟΙΩΤΗ CMOS

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός της δεύτερης άσκησης είναι η δημιουργία ενός προγράμματος για παραγωγή του επίπεδου Dataset που θα μπει ως είσοδος στην απλό προσομοιωτή που δημιουργήθηκε στην άσκηση 2. Το πρόγραμμα παράγει ένα επίπεδο Dataset που περιέχει μόνο CMOS και λαμβάνει ως είσοδο μια σχεδίαση που μπορεί να περιέχει και πύλες και CMOS. Λαμβάνει ως είσοδο ένα αρχείο ASCII με συγκεκριμένο format που περιγράφει το κύκλωμα και επιστρέφει τους κόμβους του κυκλώματος καθώς και τα rails που θα χρησιμοποιηθούν μαζί με τα set εισόδου που θέλουμε να προσομοιώσουμε.

Λειτουργία του Προγράμματος

Ο τρόπος λειτουργίας του προγράμματος είναι αρκετά απλός. Ξεκινάει διαβάζοντας το αρχείο εισόδου γραμμή προς γραμμή κάνοντας χρήση της συνάρτησης getline() και σπάει το αρχείο σε κομμάτια με χρήση της συνάρτησης strtok(). Όλα τα χρήσιμα μέρη αποθηκεύονται σε μεταβλητές για να χρησιμοποιηθούν αργότερα. Δημιουργούνται διάφοροι πίνακες που μας βοηθούν να παράξουμε σωστά το επίπεδο Dataset και να αφαιρέσουμε κόμβους που δε χρειαζόμαστε. Στο τέλος μας δίνει το Dataset σε ένα καινούργιο αρχείο. Αν όλα είναι σωστά μας δίνει μήνυμα επιτυχίας αλλιώς το πρόγραμμα κλείνει.

Ανάλυση του Κώδικα

Αρχικοποιήσεις συναρτήσεων και μεταβλητών:

```
//Functions used
//Lab3 Functions
void createCMOSNetlist(FILE* fpIn,FILE* fpOut,FILE* fplib);
void createConnections(FILE* fpIn,FILE* fpOut,FILE *fplib);
void getGateNet(FILE* fplib,int* gateInputs,int numOfGateInputs,int* gateOutputs,int numOfGateOutputs);
void updateConnections(int* gateInputs,int numOfGateInputs,int* gateOutputs,int numOfGateOutputs);
void nodeCompaction();
void printNetlist(FILE* fpOut);

//Lab2 Functions
void checkStim(int argc, char *argv[]);//checks if a stim file is entered
void getCircuitinfo(FILE* fpIn,FILE* fpOut);//gets the info from the stim file
void showHelp();//Shows help message

//Memory allocations for Circuit Data.
int numOfInputs,numOfOutputs,netlistRows,netlistCols,numOfTestIn,numOfTestOut,Unum,numOfCMOS=0,numOfConnections;
int numOfVCC = 0, numOfGND = 0;
int* VCC;
int* GND;
int* inputs;
int* finalInputs;
int* finalOutput;
int* outputs;
int **netlist;
int **connections;
int * testIn;
int * testOut;
int count,count1;
```

Η main συνάρτηση:

```
int main(int argc, char *argv[])
{
    //Check if circuit file is present
    checkStim(argc, argv);
    //open circuit file
    FILE* fpIn = fopen(argv[1],"r");
    if (fpIn == NULL)
    {
        printf("Could not find file");
    }
    char* fileOut=NULL;
    char dot = '.';
    fileOut = strtok(argv[1],&dot);
    strcat(fileOut,"_out.txt");
    FILE *fpOut = fopen(fileOut,"w");
    //get circuit file info
    getCircuitinfo(fpIn,fpOut);
    return 0;
}
```

```c
void getCircuitinfo(FILE* fpIn,FILE* fpOut)
{
    char* line=NULL;
    size_t len=0;
    ssize_t read;
    FILE* fplib = NULL;
    char* tokenOut;
    char* tokenIn;
    char space = ' ';
    char comaSemiColon[] = "',',';'";
    //Read file line by line
    while((read = getline(&line, &len, fpIn)) != -1)
    {
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        if(strcmp(line, "## LIBRARY")==0)
        {
            if ((read = getline(&line, &len, fpIn)) == -1)
            {
                printf("Problem");
                fprintf(fpOut,"Problem");
                exit(1);
            }
                line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
                fplib = fopen(line,"r");
                if (fplib == NULL)
                {
                    printf("Could not find file");
                }
        }
```

Ψήγμα κώδικα από της συνάρτηση getCircuitInfo:

Όπως βλέπουμε η συνάρτηση τραβάει μια γραμμή από το αρχείο και στη συνέχεια τη χωρίζει σε κομμάτια τα οποία είναι χρήσιμα και τα αποθηκεύει στις αντίστοιχες μεταβλητές. Η αποθήκευση γίνεται με χρήση της συνάρτησης atoi() και έτσι όλες οι μεταβλητές περιέχουν integers. Αυτό είναι ιδιαίτερα χρήσιμο για τις πράξεις που θα χρειαστούν στην συνέχεια και μας δίνει τη δυνατότητα να αφαιρούμε όλα τα κενά που υπάρχουν στο αρχείο αφού αυτό το κάνει η atoi() με επιτυχία. Ακόμη έγινε επιλογή της χρήσης integers καθώς αν γινόταν χρήση uint8_t δηλαδή unsigned integer 8 bits δεν θα μπορούσαμε να κάνουμε πράξεις για κυκλώματα με κόμβους που να έχουν αριθμό μεγαλύτερο του 256. Αφού λοιπόν οι αριθμοί σύμφωνα με την εκφώνηση είναι μέχρι και την τιμή 999 λήφθηκε η απόφαση για χρήση int. Επίσης καθώς το πρόγραμμα δεν έχει κάποια απαίτηση χαμηλής χρήσης μνήμης δεν δόθηκε ιδιαίτερη προσοχή για την ελαχιστοποίηση των πόρων όπως θα γινόταν σε κάποιο πρόγραμμα για χρήση σε κάποιο ενσωματωμένο.

Σε αυτό το σημείο όταν διαβάσουμε τη γραμμή ## LIBRARY διαβάζουμε την επόμενη γραμμή που έχει το όνομα του αρχείου που περιέχει τη βιβλιοθήκη των πυλών, ανοίγουμε το αρχείο και αποθηκεύουμε τον file pointer του αρχείο αφού θα τον χρειαστούμε αργότερα.

Η συνάρτηση getCircuitInfo λειτουργεί με τον ίδιο τρόπο για όλο το αρχείο εισόδου, εκτός κάποιον εξαιρέσεων που θα εξηγηθούν παρακάτω, οπότε δεν χρειάζεται να αναλυθεί παραπάνω.

Ψήγμα κώδικα από το διάβασμα του Netlist:

```
//Find netlist values and save them
if(strcmp(line, "## NETLIST")==0)
{
    createCMOSNetlist(fpIn,fpOut,fplib);
}
```

Για το διάβασμα του Netlist καλούμε την συνάρτηση createCMOSNetlist.

Ο πίνακας που αποθηκεύονται οι τιμές είναι NxM και γίνεται μια αντιστοιχία της λέξης “PMOS” με την τιμή ‘0’ και της λέξης “NMOS” με τη τιμή ‘1’. Δημιουργείται ένας πίνακας με μορφή:

PMOS/NMOS	GATE	SOURCE/DRAIN	DRAIN/SOURCE
0/1	Gate node	Source node/Drain Node	Drain node/Source node

Για κάθε ένα καινούργιο CMOS προστίθεται και μια γραμμή στον πίνακα.

Ψήγμα κώδικα από τη συνάρτηση getCircuitInfo() για τις περιπτώσεις TEST_IN και TEST_OUT:

```
//Find test in values and save them
if (strcmp(line, "## TEST_IN")==0)
{
    if ((read = getline(&line, &len, fpIn)) == -1)
    {
        printf("Problem");
        fprintf(fpOut,"Problem");
        exit(1);
    }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        tokenIn = strtok(line, comaSemiColon);
        testIn = (int*)malloc(sizeof(int));
        testIn[0] = atoi(tokenIn);
        count = 1;
        while (1)
        {
            tokenIn = strtok(NULL,comaSemiColon);
            if (tokenIn == NULL)
            {
                numOfTestIn = count;
                break;
            }
            count++;
            testIn = realloc(testIn,sizeof(int)*count);
            testIn[count-1] = atoi(tokenIn);
        }
    printf("## TEST_IN\n");
    fprintf(fpOut,"## TEST_IN\n");
    for (int i = 0; i < numOfTestIn; i++)
    {
        bool printed =false;
        for (int j = 0; j < numOfInputs; j++)
        {
            if (testIn[i]==inputs[j])
            {
                printed = true;
                printf("%d;",finalInputs[j]);
                fprintf(fpOut,"%d;",finalInputs[j]);
            }
        }
        if (!printed)
        {
            printf("%d;",testIn[i]);
            fprintf(fpOut,"%d;",testIn[i]);
        }
    }
```

Στη περίπτωση του TEST IN και TEST OUT διαβάζουμε την επόμενη γραμμή και ελέγχουμε αν οι είσοδοι ή έξοδοι είναι ίδιοι με κάποιο από τα inputs ή outputs, αντίστοιχα . Επειδή τα Inputs τα έχουμε αλλάξει φροντίζουμε και στα TEST IN και TEST OUT να δώσουμε τις νέες του τιμές.

Ψήγμα κώδικα από τη συνάρτηση getCircuitInfo() για τις υπόλοιπες περιπτώσεις:

```
//Find test vectors values and save them
if (strcmp(line, "## TEST_VECTORS")==0)
{
    printf("%s\n",line);
    fprintf(fpOut,"%s\n",line);
    read = getline(&line, &len, fpIn);
    printf("%s",line);
    fprintf(fpOut,"%s",line);
    read = getline(&line, &len, fpIn);
    printf("%s",line);
    fprintf(fpOut,"%s",line);
}
//Check if test ended succesfully
if (strcmp(line, "## END_TEST")==0)
{
    printf("%s\n",line);
    fprintf(fpOut,"%s\n",line);
}
//Check if test ended succesfully
if (strcmp(line, "## END_SIMULATION")==0)
{
    printf("%s\n",line);
    fprintf(fpOut,"%s\n",line);
}
if (strcmp(line, "## TESTBENCH")==0)
{
    printf("%s\n",line);
    fprintf(fpOut,"%s\n",line);
}
```

Σε αυτό το σημείο ο κώδικας απλά διαβάζεις τις γραμμές και τις εκτυπώνει αυτούσιες στο αρχείο εξόδου αφού δεν χρειάζεται να προσθέσουμε ή να αλλάξουμε κάτι.

Ψήγμα κώδικα από τη συνάρτηση createCMOSNetlist():

```
if (tokenIn[0] == 'G')
{
    Unum++;
    tokenIn = strtok(NULL, delim);
    char gatestr[100];
    fseek(fplib, 0, SEEK_SET);
    sprintf(gatestr , "## GATE %s\0",tokenIn);
    numOfGateInputs=0;
    numOfGateOutputs=0;
    strtok(NULL,delim);
    while ((in = strtok(NULL,delim))!=NULL)
    {
        if (strcmp(in,"OUT")==0)
        {
            break;
        }
        numOfGateInputs++;
        if (numOfGateInputs == 1)
        {
            gateInputs = (int*)malloc(sizeof(int)*numOfGateInputs);
            gateInputs[ numOfGateInputs-1]=atoi(in);
        }
        else
        {
            gateInputs = (int*)realloc(gateInputs,sizeof(int)*numOfGateInputs);
            gateInputs[numOfGateInputs-1]=atoi(in);
        }
    }
    while ((out=strtok(NULL,delim))!=NULL)
    {
        numOfGateOutputs++;
        if(numOfGateOutputs==1)
        {
            gateOutputs = (int*)malloc(sizeof(int)*numOfGateOutputs);
            gateOutputs[numOfGateOutputs-1]=atoi(out);
        }
        else
        {
            gateOutputs = (int*)realloc(gateOutputs,sizeof(int)*numOfGateOutputs);
            gateOutputs[numOfGateOutputs-1]=atoi(out);
        }
    }
        updateConnections(gateInputs,numOfGateInputs,gateOutputs,numOfGateOutputs);
    while((read = getline(&line, &len, fplib)) != -1)
    {
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        if(strcmp(line, gatestr)==0)
        {
            getGateNet(fplib,gateInputs,numOfGateInputs,gateOutputs,numOfGateOutputs);
            break;
        }
        else if (strcmp(line, "## END LIBRARY")==0)
        {
            printf("Could not find gate");
            fprintf(fpOut,"Could not find gate");
            exit(1);
        }
    }
```

Εδώ ο κώδικας διαβάζει το netlist που έχουμε ορίσει στο αρχείο εισόδου και περιέχει τις πύλες. Αρχικά προσπαθεί να προσδιορίσει αν αναφερόμαστε σε μια πύλη ή σε ένα CMOS. Σύμφωνα με το format εισόδου ελέγχουμε αν ο πρώτος χαρακτήρας της γραμμής είναι “G” , τότε συμπεριφερόμαστε στη γραμμή σας πύλη αλλιώς αν διαβάσουμε το U της συμπεριφερόμαστε σαν CMOS.

Για κάθε γραμμή που διαβάζουμε αυξάνουμε και ένα μετρητή με το όνομα Unum που ουσιαστικά αποτελεί το offset με το οποίο κάνουμε προσαύξηση στους κόμβους που θα αποθηκεύσουμε στο νέο Netlist.

Όταν η γραμμή έχει χαρακτηριστεί ως πύλη αρχικά αποθηκεύουμε σε μια λίστα τους κόμβους εισόδου και σε μια άλλη του κόμβους εξόδους της. Στη συνέχεια διαβάζουμε το όνομα της πύλης και δημιουργούμε ένα string με τη μορφή ## GATE [όνομα πύλης]. Με αυτό το string κάνουμε αναζήτηση στο αρχείο βιβλιοθήκης πυλών

μέχρι να βρούμε τη συγκεκριμένη γραμμή ή να βρούμε τη γραμμή ## END_LIBRARY όπου και σταματάμε και επιστρέφουμε μήνυμα λάθους.

Αν βρούμε τη πύλη που ψάχνουμε ξεκινάμε να διαβάζουμε το netlist της καλώντας την συνάρτηση getGateNet().

Πριν την κλήση της παραπάνω συνάρτησης δημιουργούμε και ενημερώνουμε έναν πίνακα connections που περιέχει όλες τις συνδέσεις του κυκλώματος. Στην πρώτη του στήλη έχει τους κόμβους του αρχείου εισόδου και στις υπόλοιπες στήλες μπαίνουν οι κόμβοι του κυκλώματος που ισοδυναμούν με αυτό τον κόμβο.

Έτσι αποκτούμε ένα πίνακα με τη παρακάτω μορφή:

Αρχικός κόμβος	Ισοδύναμοι Κόμβοι	Ισοδύναμοι Κόμβοι
1	12	25
2	14	0
5	16	0

Ο πίνακας ξεκινάει αρχικοποιημένος με μηδενικά. Έτσι για να δούμε ένας κόμβος, για παράδειγμα ο ‘2’ ,πόσους ισοδύναμους κόμβους έχει αρκεί να διασχίσουμε την γραμμή που έχει πρώτη στήλη ‘2’ μέχρι να βρούμε ‘0’.

Χρησιμοποιώντας τη συνάρτηση updateConnections δημιουργούμε και ενημερώνουμε τον παραπάνω πίνακα ώστε να έχει μοναδικές τιμές στην πρώτη του στήλη.

Για τη περίπτωση που η γραμμή ανήκει σε CMOS:

```
if (tokenIn[0] == 'U')
{
    Unum++;
    numOfCMOS++;
    if (numOfCMOS==1)
    {
        netlist = (int**)malloc(sizeof(int));//allocate memory for double pointer row
        netlist[0] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
    }
    else
    {
        netlist = (int**)realloc(netlist,sizeof(int)*numOfCMOS);//allocate memory for double pointer row
        netlist[numOfCMOS-1] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
    }
    tokenIn = strtok(NULL, delim);
    if (strcmp(tokenIn, "PMOS")==0)
    {
        netlist[numOfCMOS-1][count] = 0;
    }
    else if (strcmp(tokenIn, "NMOS")==0)
    {
        netlist[numOfCMOS-1][count] = 1;
    }
    else
    {
        printf("error");
        exit(1);
```

```
while (1)
{
    tokenIn = strtok(NULL,delim);
    if (tokenIn == NULL)
    {
        netlistRows = numOfCMOS;
        netlistCols = count+1;
        break;
    }
    count++;
    sprintf(val,"%d%s",Unum,tokenIn);
    bool inputExists=false;
    int row =0;
    for (row = 0; row < numOfConnections; row++)
    {
        if (connections[row][0]==atoi(tokenIn))
        {
            inputExists = true;
            break;
        }
    }
    if (!inputExists)
    {   numOfConnections++;
        if (numOfConnections == 1)
        {
            connections = (int**)malloc(sizeof(int));
            connections[0]= calloc(20,sizeof(int));
        }
        else
        {
            connections = (int**)realloc(connections,sizeof(int)*numOfConnections);
            connections[numOfConnections-1]= calloc(20,sizeof(int));
        }
        connections[row][0]=atoi(tokenIn);
        connections[row][1]=atoi(val);
    }
    else
    {
        int col =0;
        while (connections[row][col]!=0)
        {
            col++;
        }
        connections[row][col]=atoi(val);
    }
```

```
for (int i = 0; i < numOfInputs; i++)
{
    if (finalInputs[i]==atoi(tokenIn))
    {
        finalInputs[i]=atoi(val);
    }
}
for (int i = 0; i < numOfOutputs; i++)
{
    if (finalOutput[i]==atoi(tokenIn))
    {
        finalOutput[i]=atoi(val);
    }
}
netlist[numOfCMOS-1][count] = atoi(val);
```

Τότε διαβάζουμε κανονικά το netlist του και το αποθηκεύουμε στο νεο netlist μαζί με το offset του που έχουμε αυξήσει. Ελέγχουμε πάλι αν οι κόμβοι ανήκουν σε κάποια σύνδεση ή αν κάποιος από τους κόμβους τους αποτελεί είσοδο ή έξοδο του κυκλώματος.

Η συνάρτηση getGateNet():

```
while(((read = getline(&line, &len, fplib)) != -1) && (gateDone==false))
{
    line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
    //Find VCC and GND values and save them
    if(strcmp(line, "## RAILS")==0)
    {
        if((read = getline(&line, &len, fplib)) == -1)
        {
            printf("Problem");
            exit(1);
        }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        tokenOut = strtok(line, comaSemiColon);
        while(tokenOut != NULL)
        {
            tokenIn = strdup(tokenOut);
            tokenOut = strtok(NULL, comaSemiColon);
            tokenIn = strtok(tokenIn, &space);
            if (strcmp(tokenIn, "VCC")==0)
            {
                numOfVCC++;
                tokenIn = strtok(NULL, &space);
                sprintf(val,"%d%s",Unum,tokenIn);
                if (numOfVCC == 1)
                {
                    VCC = (int*)malloc(sizeof(int)*numOfVCC);
                    VCC[numOfVCC-1] = atoi(val);
                }
                else
                {
                    VCC = (int*)realloc(VCC,sizeof(int)*numOfVCC);
                    VCC[numOfVCC-1] = atoi(val);
                }

            }
            else if (strcmp(tokenIn, "GND")==0)
            {
                numOfGND++;
                tokenIn = strtok(NULL, &space);
                sprintf(val,"%d%s",Unum,tokenIn);
                if (numOfGND == 1)
                {
                    GND = (int*)malloc(sizeof(int)*numOfGND);
                    GND[numOfGND-1] = atoi(val);
                }
                else
                {
                    GND = (int*)realloc(GND,sizeof(int)*numOfGND);
                    GND[numOfGND-1] = atoi(val);
                }
            }
```

```
    //Find input values and save them
    if(strcmp(line, "## INPUTS")==0)
    {
        if ((read = getline(&line, &len, fplib)) == -1)
        {
            printf("Problem");
            exit(1);
        }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0
        tokenIn = strtok(line, comaSemiColon);
        while (tokenIn != NULL)
        {
            numOfIn++;
            bool inputExists = false;
            int row=0;
            for (row = 0; row < numOfConnections; row++)
            {
                if (connections[row][0]==gateInputs[numOfIn-1])
                {
                    inputExists = true;
                    break;
                }
            }
            int col =0;
            while (connections[row][col]!=0)
            {
                col++;
            }

            if (inputExists)
            {
                sprintf(val,"%d%s",Unum,tokenIn);
                connections[row][col]=atoi(val);
            }
            for (int i = 0; i < numOfInputs; i++)
            {
                if (finalInputs[i]==gateInputs[numOfIn-1])
                {
                    finalInputs[i] = atoi(val);
                    break;
                }
            }
            tokenIn = strtok(NULL,delim);
        }
    }
```

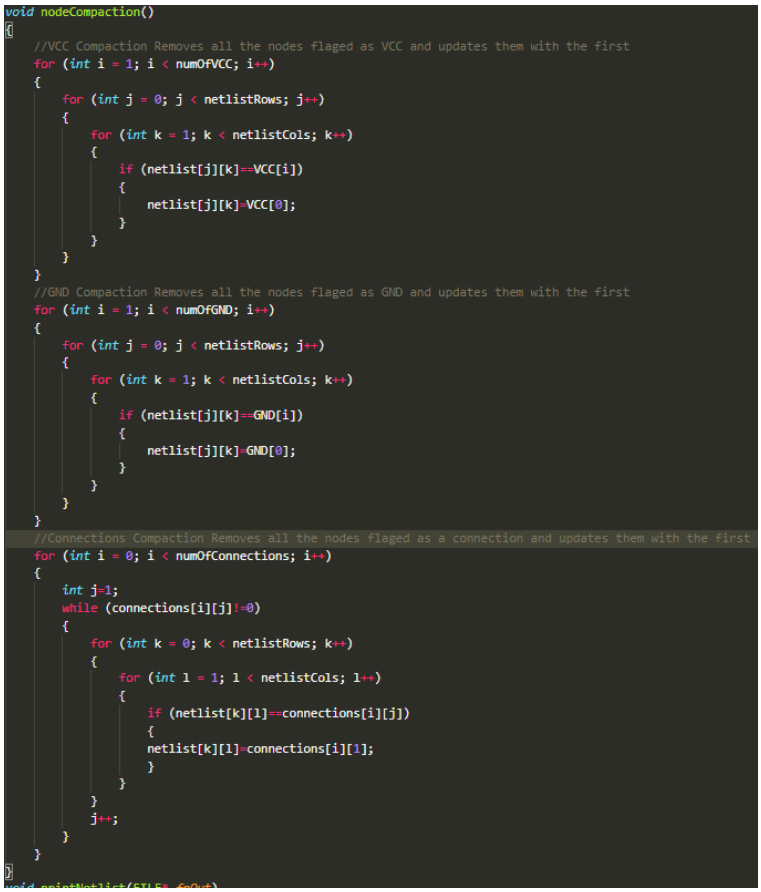
```
    //Find output values and save them
    if(strcmp(line, "## OUTPUTS")==0)
    {
        if ((read = getline(&line, &len, fplib)) == -1)
        {
            printf("Problem");
            // fprintf(fpOut,"Problem");
            exit(1);
        }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        tokenIn = strtok(line, comaSemiColon);
        while (tokenIn != NULL)
        {
            numOfOut++;
            bool inputExists = false;
            int row=0;
            for (row = 0; row < numOfConnections; row++)
            {
                if (connections[row][0]==gateOutputs[numOfOut-1])
                {
                    inputExists = true;
                    break;
                }
            }
            int col =0;
            while (connections[row][col]!=0)
            {
                col++;
            }

            if (inputExists)
            {
                sprintf(val,"%d%s",Unum,tokenIn);
                connections[row][col]=atoi(val);
            }
            for (int i = 0; i < numOfInputs; i++)
            {
                if (finalOutput[i]==gateOutputs[numOfOut-1])
                {
                    finalOutput[i] = atoi(val);
                    break;
                }
            }
            tokenIn = strtok(NULL,delim);
        }
```

```
if(strcmp(line, "## NETLIST")==0)
{
    while (1)
    {
        count=0;
    if ((read = getline(&line, &len, fplib)) == -1)
    {
        printf("Problem");
        // fprintf(fpOut,"Problem");
        exit(1);
    }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        if (strcmp(line, "## END_GATE")==0)
        {
            gateDone = true;
            break;
        }
        numOfCMOS++;
        if (numOfCMOS==1)
        {
            netlist = (int**)malloc(sizeof(int));//allocate memory for double pointer row
            netlist[0] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
        }
        else
        {
            netlist = (int**)realloc(netlist,sizeof(int)*(numOfCMOS));//allocate memory for double pointer row
            netlist[numOfCMOS-1] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
        }
        tokenIn = strtok(line, &space);
        tokenIn = strtok(NULL, &space);
        if (strcmp(tokenIn, "PMOS")==0)
        {
            netlist[numOfCMOS-1][count] = 0;
        }
        else if (strcmp(tokenIn, "NMOS")==0)
        {
            netlist[numOfCMOS-1][count] = 1;
        }
        else
        {
            printf("error");
            exit(1);
        }
```

```
while (1)
{
    tokenIn = strtok(NULL,&space);
    if (tokenIn == NULL)
    {
        netlistRows = numOfCMOS;
        netlistCols = count+1;
        break;
    }
    count++;
    sprintf(val,"%d%s",Unum,tokenIn);
    netlist[numOfCMOS-1][count] = atoi(val);
}
```

Διαβάζει τα στοιχεία του της πύλης από τη βιβλιοθήκη και εισάγει στο νέο netlist τις τιμές του προσαυξημένες με το offset τους. Ελέγχει τους κόμβους του κυκλώματος ώστε να ενημερωθεί σωστά ο πίνακας των connections και οι είσοδοι και έξοδοι του κυκλώματος.

Η συνάρτηση nodeCompaction():



```
void nodeCompaction()
{
    //VCC Compaction Removes all the nodes flaged as VCC and updates them with the first
    for (int i = 1; i < numOfVCC; i++)
    {
        for (int j = 0; j < netlistRows; j++)
        {
            for (int k = 1; k < netlistCols; k++)
            {
                if (netlist[j][k]==VCC[i])
                {
                    netlist[j][k]=VCC[0];
                }
            }
        }
    }
    //GND Compaction Removes all the nodes flaged as GND and updates them with the first
    for (int i = 1; i < numOfGND; i++)
    {
        for (int j = 0; j < netlistRows; j++)
        {
            for (int k = 1; k < netlistCols; k++)
            {
                if (netlist[j][k]==GND[i])
                {
                    netlist[j][k]=GND[0];
                }
            }
        }
    }
    //Connections Compaction Removes all the nodes flaged as a connection and updates them with the first
    for (int i = 0; i < numOfConnections; i++)
    {
        int j=1;
        while (connections[i][j]!=0)
        {
            for (int k = 0; k < netlistRows; k++)
            {
                for (int l = 1; l < netlistCols; l++)
                {
                    if (netlist[k][l]==connections[i][j])
                    {
                    netlist[k][l]=connections[i][1];
                    }
                }
            }
            j++;
        }
    }
}
```

Η συνάρτηση αυτή έχει μια πολύ απλή διαδικασία να εκτελέσει. Διαβάζει τους κόμβους του κυκλώματος και αφαιρεί τους αχρείαστους κόμβους που γνωρίζουμε ότι είναι VCC ή GND ή γνωστές συνδέσεις.

Έτσι διαβάζει ένα ένα το νεο Netlist και από τις λίστες με τα αποθηκευμένα VCC , GND ενημερώνει κάθε κόμβο στη πρώτη τιμή της λίστας.

Για παράδειγμα αν έχουμε μια λίστα που έχει ως VCC τις τιμές 1, 13 ,56. Όπου διαβάζει στο netlist τις τιμές 13, 56 τις αλλάζει σε 1.

Για τις συνδέσεις κάνουμε την ίδια διαδικασία αλλά αντί να βάλουμε τη πρώτη τιμή που είναι η τιμή του εξωτερικού κόμβου βάζουμε τη δεύτερη τιμή που είναι ουσιαστικά η πρώτη τιμή που έχουμε προσθέσει στο netlist προσαυξημένη με το offset.

Η συνάρτηση printNetlist():

```
void printNetlist(FILE* fpOut)
{
    printf("## RAILS\n");
    fprintf(fpOut,"## RAILS\n");
    printf("VCC %d ; GND %d\n",VCC[0],GND[0]);
    fprintf(fpOut,"VCC %d ; GND %d\n",VCC[0],GND[0]);
    printf("## INPUTS\n");
    fprintf(fpOut,"## INPUTS\n");
    for (int i = 0; i < numOfInputs; i++)
    {
        if (i==(numOfInputs-1))
        {
            printf("%d\n",finalInputs[i]);
            fprintf(fpOut,"%d\n",finalInputs[i]);
            break;
        }

        printf("%d,",finalInputs[i]);
        fprintf(fpOut,"%d,",finalInputs[i]);
    }
    printf("## OUTPUTS\n");
    fprintf(fpOut,"## OUTPUTS\n");
    for (int i = 0; i < numOfOutputs; i++)
    {
        if (i==(numOfOutputs-1))
        {
            printf("%d\n",finalOutput[i]);
            fprintf(fpOut,"%d\n",finalOutput[i]);
            break;
        }

        printf("%d,",finalOutput[i]);
        fprintf(fpOut,"%d,",finalOutput[i]);
    }
    printf("## NETLIST\n");
    fprintf(fpOut,"## NETLIST\n");
    int num=0;
    for (int i = 0; i < netlistRows; i++)
    {
        num++;
        if (netlist[i][0]==1)
        {
            printf("U%d PMOS ",num);
            fprintf(fpOut,"U%d PMOS ",num);
        }
        else
        {
            printf("U%d NMOS ",num);
            fprintf(fpOut,"U%d NMOS ",num);
        }

        for (int j = 1; j < netlistCols; j++)
        {
            printf("%d ",netlist[i][j]);
            fprintf(fpOut,"%d ",netlist[i][j]);
        }
        printf("\n");
        fprintf(fpOut,"\n");
    }
```

Μια πολύ απλή συνάρτηση που εκτυπώνει το τελικό Netlist στο αρχείο εξόδου.

Ανάλυση της Διαδικασίας

Παράδειγμα με χρήση του κυκλώματος που δόθηκε για την πύλη AND.

```
##RAILS

## INPUTS
1, 2
## OUTPUTS
3
## NETLIST
G1, NOT, IN,1, OUT, 4
G2, NOT, IN, 2, OUT 5
G3, NOR_2,  IN,  4,  5,  OUT, 3
## TESTBENCH
## TEST_IN
1 ; 2
## TEST_OUT
3
## TEST_VECTORS
0 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 1
## SIMULATE
## TEST_VECTORS
1 ; 0
## SIMULATE
## TEST_VECTORS
1 ; 1
## SIMULATE
## END_TEST
## END_SIMULATION
```

```
## GATE NOT
## RAILS
VCC 1 ; GND 4
## INPUTS
2
## OUTPUTS
3
## NETLIST
U1 PMOS 2 1 3
U2 NMOS 2 3 4
## END_GATE

## GATE NOR_2
## RAILS
VCC 1 ; GND 6
## INPUTS
2 ; 3
## OUTPUTS
5
## NETLIST
U1 PMOS 2 1 4
U2 PMOS 3 4 5
U3 NMOS 2 5 6
U4 NMOS 3 5 6
## END_GATE

##END LIBRARY
```

Από τη διαδικασία παράγονται οι παρακάτω πίνακες:

VCC:

11	21	31

GND:

14	24	36

Οι παραπάνω πίνακες γεμίζουν κάθε φορά με τιμές από τους κόμβους της βιβλιοθήκης της πύλης και έχουν το offset της πύλης.

Από τη τιμή που έχουν αρχικά απανογράφονται με τη τιμή που έχουν στο τέλος αυτό το δέιχνουμε με το ‘->’

Inputs:

1 -> 12	2 -> 22

Outputs:

3 -> 35

Οι παραπάνω πανωγράφονται με το Input μια πύλης μαζί με το offset αν αυτή η τιμή του input ή output είναι και τιμή input ή output του αρχικού κυκλώματος.

Connections:

1	12	0
4	13	32
2	22	0
5	23	33
3	35	0

Ο πίνακας Connections δημιουργείται κατάλληλα για τη συσχέτιση κόμβων με τη τους αρχικούς κόμβους του κυκλώματος.

Νέο NETLIST:

0	12	11	13
1	12	13	14
0	22	21	13
1	22	23	24
0	13	31	34
0	33	34	35
1	13	35	14
1	33	35	14

Το Netlist παράγεται από τους κόμβους της βιβλιοθήκης για μια συγκεκριμένη πύλη μαζί με το offset της.

NETLIST μετά το Compaction:

0	12	11	13
1	12	13	14
0	22	11	13
1	22	23	14
0	13	11	34
0	23	34	35
1	13	35	36
1	23	35	36

Ο τελικός πίνακας που είναι ουσιαστικά και το αποτέλεσμα που θα εκτυπώσουμε κατάλληλα ώστε να έχει το σωστό format για τον απλό προσομοιωτή.

Τέλος δημιουργούμε σωστά τις υπόλοιπες γραμμές τους αρχείου που περιέχουν την πληροφορία για τα TEST IN , TEST OUT και τα VECTORS εισόδου.

Παράδειγμα εκτύπωσης του προγράμματος για το παραπάνω παράδειγμα

```
## RAILS
VCC 11 ; GND 14
## INPUTS
12,22
## OUTPUTS
35
## NETLIST
U1 NMOS 12 11 13
U2 PMOS 12 13 14
U3 NMOS 22 11 23
U4 PMOS 22 23 14
U5 NMOS 13 11 34
U6 NMOS 23 34 35
U7 PMOS 13 35 14
U8 PMOS 23 35 14
## TESTBENCH
## TEST_IN
12;22;
## TEST_OUT
35;
## TEST_VECTORS
0 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 1
## SIMULATE
## TEST_VECTORS
1 ; 0
## SIMULATE
## TEST_VECTORS
1 ; 1
## SIMULATE
## END_TEST
```

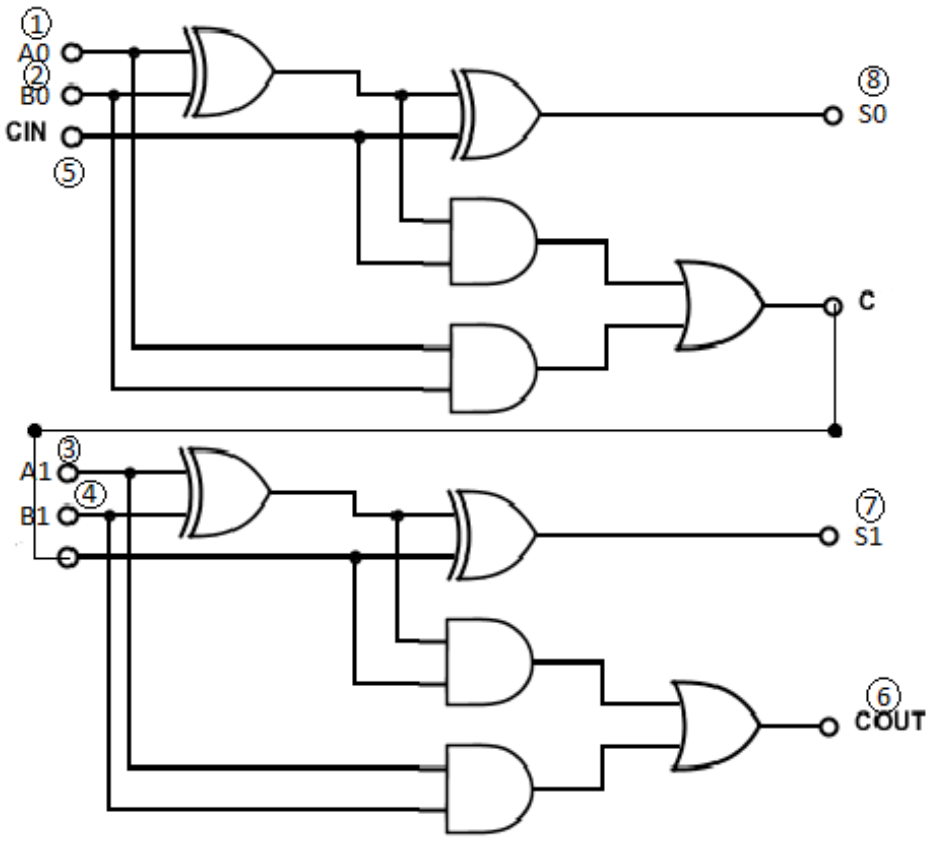
Ripple Carry Adder 2 bits

Το κύκλωμα που χρησιμοποιήθηκε ως αρχείο εισόδου για το πρόγραμμα της άσκησης 3 είναι το παρακάτω.

Έχει ως εισόδους τα pins 1,2,3,4,5. Ως έξοδοι είναι τα pins 6,7,8.

Ο αριθμός 0 εισάγεται στα pins 1,2 , ο αριθμός 1 εισάγεται στα 3,4 και το Cin είναι το 5.

Στην έξοδο το pin 6 είναι το Cout, το pin 7 είναι το S1 και το pin 8 είναι το S0.




```
## LIBRARY
RCA.LIB
## RAILS

## INPUTS
1, 2, 3, 4, 5
## OUTPUTS
6 ; 7 ; 8
## NETLIST
G1 XOR_2 ; IN 2,4 ; OUT 9
G2 XOR_2 ; IN 9,5 ; OUT 8
G3 AND_2 ; IN 9,5 ; OUT 10
G4 AND_2 ; IN 2,4 ; OUT 11
G5 OR_2 ; IN 10,11 ; OUT 12
G6 XOR_2 ; IN 1,3 ; OUT 13
G7 XOR_2 ; IN 13,12 ; OUT 7
G8 AND_2 ; IN 13,12 ; OUT 14
G9 AND_2 ; IN 1,3 ; OUT 15
G10 OR_2 ; IN 14,15 ; OUT 6
## TESTBENCH
## TEST_IN
1, 2, 3, 4, 5
## TEST_OUT
6 ; 7 ; 8
## TEST_VECTORS
0 ; 0 ; 0 ; 0 ; 0
## SIMULATE
```

Έγινε εξαντλητική προσομοίωση και για τα 32 vectors εισόδου και επιβεβαιώθηκε η ορθή λειτουργία τόσο του προγράμματος της άσκησης 3 όσο και της άσκησης 2.

Τα αρχεία εισόδου και εξόδου βρίσκονται στον ίδιο φάκελο με αυτό το έγγραφο με ονόματα:

RCA_2bit.txt : Το αρχείο εισόδου της άσκησης 3.

RCA.LIB : Το αρχείο βιβλιοθήκης για της πύλες του αρχείου εισόδου.

RCA_2bit_out.txt : Το αρχείο εξόδου της άσκησης 3.

Παρακάτω βλέπουμε το αρχείο εξόδου της άσκησης 3 με όνομα **RCA_2bit_out.txt.**

```
## RAILS
VCC 11 ; GND 17
## INPUTS
612,112,613,113,213
## OUTPUTS
104,74,24
## NETLIST
U1 PMOS 112 11 12
U2 PMOS 19 11 13
U3 PMOS 18 12 14
U4 PMOS 113 13 14
U5 NMOS 112 14 15
U6 NMOS 110 14 16
U7 NMOS 113 15 17
U8 NMOS 111 16 17
U9 PMOS 113 11 18
U10 NMOS 113 18 17
U11 PMOS 112 11 19
U12 NMOS 112 19 17
U13 PMOS 112 11 110
U14 NMOS 112 110 17
U15 PMOS 113 11 111
U16 NMOS 113 111 17
U17 PMOS 14 11 22
U18 PMOS 29 11 23
U19 PMOS 28 22 24
U20 PMOS 213 23 24
U21 NMOS 14 24 25
U22 NMOS 210 24 26
U23 NMOS 213 25 17
U24 NMOS 211 26 17
U25 PMOS 213 11 28
U26 NMOS 213 28 17
U27 PMOS 14 11 29
U28 NMOS 14 29 17
U29 PMOS 14 11 210
U30 NMOS 14 210 17
U30 NMOS 14 210 17
U31 PMOS 213 11 211
U32 NMOS 213 211 17
U33 PMOS 14 11 32
U34 PMOS 213 11 32
U35 NMOS 14 32 33
U36 NMOS 213 33 17
U37 PMOS 32 11 37
U38 NMOS 32 37 17
U39 PMOS 112 11 42
U40 PMOS 113 11 42
U41 NMOS 112 42 43
U42 NMOS 113 43 17
U43 PMOS 42 11 47
U44 NMOS 42 47 17
U45 PMOS 37 11 52
U46 PMOS 47 52 53
U47 NMOS 47 53 17
U48 NMOS 37 53 17
U49 PMOS 53 11 54
U50 NMOS 53 54 17
U51 PMOS 612 11 62
U52 PMOS 69 11 63
U53 PMOS 68 62 64
U54 PMOS 613 63 64
U55 NMOS 612 64 65
U56 NMOS 610 64 66
U57 NMOS 613 65 17
U58 NMOS 611 66 17
U59 PMOS 613 11 68
U60 NMOS 613 68 17
U61 PMOS 612 11 69
U62 NMOS 612 69 17
U63 PMOS 612 11 610
U64 NMOS 612 610 17
U65 PMOS 613 11 611
U66 NMOS 613 611 17
U67 PMOS 64 11 72
U68 PMOS 79 11 73
U69 PMOS 78 72 74
U70 PMOS 54 73 74
U71 NMOS 64 74 75
U72 NMOS 710 74 76
U73 NMOS 54 75 17
U74 NMOS 711 76 17
U75 PMOS 54 11 78
U76 NMOS 54 78 17
U77 PMOS 64 11 79
U78 NMOS 64 79 17
U79 PMOS 64 11 710
U80 NMOS 64 710 17
U81 PMOS 54 11 711
U82 NMOS 54 711 17
U83 PMOS 64 11 82
U84 PMOS 54 11 82
U85 NMOS 64 82 83
U86 NMOS 54 83 17
U87 PMOS 82 11 87
U88 NMOS 82 87 17
U89 PMOS 612 11 92
U90 PMOS 613 11 92
U91 NMOS 612 92 93
U92 NMOS 613 93 17
U93 PMOS 92 11 97
U94 NMOS 92 97 17
U95 PMOS 87 11 102
U96 PMOS 97 102 103
U97 NMOS 97 103 17
U98 NMOS 87 103 17
U99 PMOS 103 11 104
U100 NMOS 103 104 17
## TESTBENCH
## TEST_IN
612;112;613;113;213;
## TEST_OUT
104;74;24;
## TEST_VECTORS
0 ; 0 ; 0 ; 0 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 0 ; 0 ; 1
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 0 ; 1 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 0 ; 1 ; 1
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 1 ; 0 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 1 ; 0 ; 1
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 1 ; 1 ; 0
## SIMULATE
## TEST_VECTORS
0 ; 0 ; 1 ; 1 ; 1
## SIMULATE
## TEST_VECTORS
0 ; 1 ; 0 ; 0 ; 0
## SIMULATE
```

Τέλος βλέπουμε μερικά αποτελέσματα αφού γίνει η προσομοίωση χρησιμοποιώντας το παραπάνω αρχείο εξόδου ως είσοδο στον απλό προσομοιωτή της άσκησης 2.

Για είσοδο 0,0,0,0,0 έχουμε αποτέλεσμα 000

```
TEST VECTORS ARE :
Input:612 = 0 Input:112 = 0 Input:613 = 0 Input:113 = 0 Input:213 = 0
SIMULATING CIRCUIT...

Number of Nodes: 61
Nodes :
11 12 13 14 15 16 17 18 19 22 23 24 25 26 28 29 32 33 37 42 43 47 52 5
Graph converged

Output Node 104 = 0

Output Node 74 = 0

Output Node 24 = 0
```

Για είσοδο 0,0,0,0,1 δηλαδή μόνο Cin=1 έχουμε αποτέλεσμα 001.

```
TEST VECTORS ARE :
Input:612 = 0 Input:112 = 0 Input:613 = 0 Input:113 = 0 Input:213 = 1
SIMULATING CIRCUIT...

Number of Nodes: 61
Nodes :
11 12 13 14 15 16 17 18 19 22 23 24 25 26 28 29 32 33 37 42 43 47 52 5
Graph converged

Output Node 104 = 0

Output Node 74 = 0

Output Node 24 = 1
```

Για είσοδο 1,1,0,1,0 δηλαδή 11+01+0 έχουμε αποτέλεσμα 100.

```
TEST VECTORS ARE :
Input:612 = 1 Input:112 = 1 Input:613 = 0 Input:113 = 1 Input:213 = 0
SIMULATING CIRCUIT...

Number of Nodes: 61
Nodes :
11 12 13 14 15 16 17 18 19 22 23 24 25 26 28 29 32 33 37 42 43 47 52
Graph converged

Output Node 104 = 1

Output Node 74 = 0

Output Node 24 = 0
```

Τέλος για είσοδο 1,1,1,1,1 δηλαδή 11+11+1 έχουμε αποτέλεσμα 111.

```
Input:612 = 1 Input:112 = 1 Input:613 = 1 Input:113 = 1 Input:213 = 1
SIMULATING CIRCUIT...

Number of Nodes: 61
Nodes :
11 12 13 14 15 16 17 18 19 22 23 24 25 26 28 29 32 33 37 42 43 47 52 5
Graph converged

Output Node 104 = 1

Output Node 74 = 1

Output Node 24 = 1
```

Συμπέρασμα

Όπως φαίνεται και από όλα τα παραπάνω αποτελέσματα το πρόγραμμα μας δουλεύει κανονικά και δεν έχει προβλήματα ούτε με πανωγραφές ούτε με διπλότυπα στις ονομασίες των κόμβων. Επίσης δοκιμάστηκε και με λάθος αρχείο εισόδου και λειτούργησε κανονικά εμφανίζοντας τους προβληματικούς κόμβους με επιτυχία.