ΕΡΓΑΣΤΗΡΙΟ 2

28/03/2021

ΕΝΑΣ ΑΠΛΟΣ ΠΡΟΣΟΜΟΙΩΤΗΣ ON/OFF ΓΙΑ ΚΥΚΛΩΜΑΤΑ CMOS

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός της δεύτερης άσκησης είναι η δημιουργία ενός απλού προσομοιωτή on/off για κυκλώματα CMOS. Ο προσομοιωτής δουλεύει για απλά κυκλώματα με 2-3 εισόδους. Λαμβάνει ως είσοδο ένα αρχείο ASCII με συγκεκριμένο format που περιγράφει το κύκλωμα CMOS και υλοποιώντας χρωματισμό γράφου όπως περιγράφηκε στις διαλέξεις επιστρέφει τις τιμές όλως των κόμβων του κυκλώματος καθώς και την τιμή της εξόδου για κάθε ένα από τα set εισόδου που δίνονται.

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. Στη συνέχεια εκτυπώνει τα χαρακτηριστικά του κυκλώματος και ξεκινάει η διαδικασία της προσομοίωσης για κάθε ένα από τα διανύσματα εισόδου που έχουν δοθεί. Εκτελεί τον χρωματισμό του γράφου και έτσι μπορούμε να γνωρίζουμε την τιμή που έχει κάθε κόμβος καθώς και επίσης αν υπάρχει κάποιο βραχυκύκλωμα μέσα στο κύκλωμα ή αν η έξοδος έχει ακαθόριστη τιμή. Στο τέλος μας δίνει τα αποτελέσματα καθώς και τα σημεία που χρειάζονται διόρθωση μέσα στο κύκλωμα. Αν όλα είναι σωστά μας δίνει μήνυμα επιτυχίας αλλιώς μήνυμα αποτυχίας.

Ανάλυση του Κώδικα

Αρχικοποιήσεις συναρτήσεων και μεταβλητών:

```
//Functions used
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 doTestBench(FILE* fpIn,FILE* fpOut);//does the TESTBENCH
void showSimulation(FILE * fpOut);//Does simulation and presents it
void showCircuitInfo(FILE * fpOut);//Shows obtained circuit info
void colorGraph(FILE* fpOut);//Does the graph coloring algorithm
int findGraphPos(int nodeVal);//Finds the position of a value in the graph
void showHelp();//Shows help message
void printGraph(FILE* fpOut);//Prints the graph

//Memory allocations for Circuit Data.
int VCC,GND,numOfInputs,numOfOutputs,netlistRows,netlistCols,numOfTestIn,numOfTestOut,numOfGraphs,numOfSimulations,correctSims;
int numOfNodes = 0;
int* inputs;
int* outputs;
int **netlist;
int* testIn;
int* testOut;
int* testVector;
int count,count1;
int* nodes;
int **graph;
```

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 = '.';
    // strcat(fileOut,argv[1]);
    fileOut = strtok(argv[1],&dot);
    strcat(fileOut,"_out.txt");
    FILE *fpOut = fopen(fileOut,"w");
    fprintf(fpOut,"Output file for file %s\n\n",argv[1]);
    //get circuit file info
    getCircuitinfo(fpIn,fpOut);
    return 0;
}
```

Ψήγμα κώδικα από της συνάρτηση getCircuitInfo:

```
void getCircuitinfo(FILE* fpIn,FILE* fpOut)
{
    char* line=NULL;
    size_t len=0;
    ssize_t read;

    char* tokenOut;
    char* tokenIn;
    char semicolon = ';';
    char space = ' ';
    //Read file line by line
    while((read = getline(&line, &len, fpIn)) != -1)
    {
        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, fpIn)) == -1)
            {
                printf("Problem");
                fprintf(fpOut,"Problem");
                exit(1);
            }
                line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
                tokenOut = strtok(line, &semicolon);
            while(tokenOut != NULL)
            {
                tokenIn = strdup(tokenOut);
                tokenOut = strtok(NULL, &semicolon);
                tokenIn = strtok(tokenIn, &space);
                if (strcmp(tokenIn, "VCC")==0)
                {
                    tokenIn = strtok(NULL, &space);
                    VCC = atoi(tokenIn);
                }
                else if (strcmp(tokenIn, "GND")==0)
                {
                    tokenIn = strtok(NULL, &space);
                    GND = atoi(tokenIn);
                }
            }
        }
```

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.

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)
{   count1=0;
    while (1)
    {
    if ((read = getline(&line, &len, fpIn)) == -1)
    {
        printf("Problem");
        fprintf(fpOut,"Problem");
        exit(1);
    }
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        if (strcmp(line, "## TESTBENCH")==0)
        {
            break;
        }
        count=0;
        if (count1==0)
        {
            netlist = (int**)malloc(sizeof(int));//allocate memory for double pointer row
            netlist[count1] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
        }
        else
        {
            netlist = (int**)realloc(netlist,sizeof(int)*(count1+1));//allocate memory for double pointer row
            netlist[count1] = (int*)malloc(sizeof(int)*4);//allocate memory for array columns
        }
        tokenIn = strtok(line, &space);
        tokenIn = strtok(NULL, &space);
        if (strcmp(tokenIn, "PMOS")==0)
        {
            netlist[count1][count] = 0;
        }
        else if (strcmp(tokenIn, "NMOS")==0)
        {
            netlist[count1][count] = 1;
        }
        else
        {
            printf("error");
            fprintf(fpOut,"error");
            exit(1);
        }
        while (1)
        {
            tokenIn = strtok(NULL,&space);
            if (tokenIn == NULL)
            {
                netlistRows = count1+1;
                netlistCols = count+1;
                break;
            }
            count++;
            netlist[count1][count] = atoi(tokenIn);
        }
        count1++;
    }
}
```

Για το διάβασμα του Netlist η διαδικασία είναι παρόμοια με τη διαφορά ότι 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 TESTBENCH και END_SIMULATION:

```
//Find Testbench values save them and do test
if(strcmp(line, "## TESTBENCH")==0)
{
    numOfSimulations=0;
    correctSims=0;
    doTestBench(fpIn,fpOut);
}
if(strcmp(line, "## END_SIMULATION")==0)//For this to work right we have to add a blank row in out text file
                                        //Because we cannot compare last line with a string because of EOF.
{
    printf("\nSIMULATION ENDED\n");
    fprintf(fpOut,"SIMULATION ENDED");
    break;
}
```

Σε αυτό το σημείο αρχικοποιούμε 2 μεταβλητές που μας βοηθούν να ελέγξουμε έχει πάει καλά η προσομοίωση και να αποφασίσουμε αν το κύκλωμα είναι σωστό ή αν σε κάποιο διάνυσμα εισόδου υπάρχει πρόβλημα, οπότε και δεν είναι σωστό το κύκλωμα.

Στη συνέχεια εκτελούμε το TestBench που δηλώνεται μετά τη λέξη TESTBENCH.

Τέλος αφού διαβάσουμε τη λέξη END_SIMULATION κλείνουμε τη προσομοίωση και τερματίζει το πρόγραμμα.

Ψήγμα κώδικα από τη συνάρτηση doTestBench():

```
//Do test
void doTestBench(FILE* fpIn,FILE* fpOut)
{
    char* line=NULL;
    size_t len=0;
    ssize_t read;
    char* token;
    char semicolon = ';';

    while ((read = getline(&line, &len, fpIn)) != -1)
    {
        line[strlen(line)-1]='\0'; // Overwrite char \n with \0.
        //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.
                token = strtok(line, &semicolon);
                testIn = (int*)malloc(sizeof(int));
                testIn[0] = atoi(token);
                count = 1;
                while (1)
                {
                    token = strtok(NULL,&semicolon);
                    if (token == NULL)
                    {
                        numOfTestIn = count;
                        break;
                    }
                    count++;
                    testIn = realloc(testIn,sizeof(int)*count);
                    testIn[count-1] = atoi(token);
                }
        }
```

Σε αυτό το σημείο ο κώδικας συνεχίζει να διαβάζει και να αποθηκεύει τα υπόλοιπα στοιχεία του κυκλώματος όπως στη συνάρτηση getCircuitInfo() ,τα οποία όμως σχετίζονται μόνο με το συγκεκριμένο TESTBENCH.

Ψήγμα κώδικα από τη συνάρτηση doTestBench() για το κομμάτι SIMULATE και END_TEST:

```
//start simulation
if (strcmp(line, "## SIMULATE")==0)
{
    //save number of simualtions
    numOfSimulations++;
    showSimulation(fpOut);
}
//Check if test ended succesfully
if (strcmp(line, "## END_TEST")==0)
{
    if (numOfSimulations == correctSims)
    {
        printf("\nTEST ENDED SUCCESFULY\n\n");
        fprintf(fpOut,"\nTEST ENDED SUCCESFULY\n\n");
        printf("Press ENTER key to continue");
        getchar();
    }
    else
    {
        printf("\nTEST ENDED WITH ERRORS\n");
        printf("\nPlease fix the error indicated above and run the test again\n\n");
        fprintf(fpOut,"\nTEST ENDED WITH ERRORS\n");
        fprintf(fpOut,"\nPlease fix the error indicated above and run the test again\n\n");
        printf("Press ENTER key to continue");
        getchar();
    }
    break;
}
```

Εδώ όταν ο κώδικας διαβάζει τη γραμμή SIMULATE καλεί τη συνάρτηση showSimulation και αυξάνει τον αριθμό των προσομοιώσεων που έχει εκτελέσει.

Όταν διαβάζει τη γραμμή END_TEST ελέγχει αν ο αριθμός των προσομοιώσεων είναι ίσος με τον αριθμό των προσομοιώσεων που δεν έχουν σφάλματα και αν είναι εκτυπώνει μήνυμα επιτυχίας αλλιώς μήνυμα λάθους.

Η συνάρτηση showSimulation():

```
//show simulation and graph coloring
void showSimulation(FILE * fpOut)
{
    showCircuitInfo(fpOut);
    printf("\nSIMULATING CIRCUIT...\n\n");
    fprintf(fpOut,"\nSIMULATING CIRCUIT...\n\n");
    colorGraph(fpOut);
    return;
}
```

Μια απλή συνάρτηση που εκτυπώνει τα χαρακτηριστικά του κυκλώματος και μετά εκτελεί το χρωματισμό του γράφου και τη προσομοίωση του κυκλώματος.

Η συνάρτηση colorGraph():

```
//do the graph coloring
void colorGraph(FILE* fpOut)
{
    int sameAsPrev = 0;
    bool nodeExists,converged;
    nodes = (int*)malloc(sizeof(int)*netlistRows*netlistCols);
    nodes[0]=netlist[0][1];
    numOfNodes=1;
    //calculating Nodes
    //get each node one time without duplicates
    for (int i = 0; i <  netlistRows; i++)
    {
        for (int j = 1; j < netlistCols; j++)
        {
            nodeExists = false;
            for (int k = 0; k < numOfNodes; k++)
            {
                if (nodes[k]==netlist[i][j])
                {
                    nodeExists = true;
                }
            }
            if (!nodeExists)
            {
                nodes[numOfNodes]=netlist[i][j];
                numOfNodes++;
            }
        }
    }
printf("Number of Nodes: %d\n",numOfNodes);
fprintf(fpOut,"Number of Nodes: %d\n",numOfNodes);

//Array sorting in ascending order
//We have low amount of nodes so we dont care about
int a;
for (int i = 0; i < numOfNodes; ++i)
    {
        for (int j = i + 1; j < numOfNodes; ++j)
        {
            if (nodes[i] > nodes[j])
            {
                a =  nodes[i];
                nodes[i] = nodes[j];
                nodes[j] = a;
            }
        }
    }

printf("Nodes :\n");
fprintf(fpOut,"Nodes :\n");
for (int i = 0; i < numOfNodes; i++)
{
    printf("%d ",nodes[i]);
    fprintf(fpOut,"%d ",nodes[i]);
}
numOfGraphs = 3;
//Allocate memory for graph
graph = (int**)malloc(sizeof(int)*(numOfGraphs));//allocate memory for graph
for (int i = 0; i < numOfGraphs; i++)
{
    graph[i] = (int*)malloc(sizeof(int)*(numOfNodes));//allocate memory for graph
}
//Populate graph with nodes
for (int i = 0; i < numOfNodes; i++)
{
    graph[0][i]=nodes[i];
}
// Populate graph with node's values for first time
for (int i = 0; i < numOfNodes; i++)
{
    if (graph[0][i]==VCC)
    {
        graph[1][i]=1;
    }
    else if (graph[0][i]==GND)
    {
        graph[1][i]=0;
    }
    else
    {
        graph[1][i]=2;
    }
    for (int j = 0; j < numOfTestIn; j++)
    {
        if (graph[0][i]==testIn[j])
        {
            graph[1][i]=testVector[j];
        }
    }
}
printGraph(fpOut);
```

Η συνάρτηση colorGraph ξεκινάει διαβάζοντας το netlist και αποθηκεύει σε μια λίστα όλους τους κόμβους που θα βρει από μια φορά, δηλαδή χωρίς διπλότυπα. Έτσι για οποιονδήποτε αριθμό κόμβων και για netlist με “κενά” μπορούμε να αποθηκεύουμε τους κόμβους ώστε να γνωρίζουμε τους κόμβους του γράφου που θα φτιάξουμε. Στη συνέχεια εκτελούμε αριθμητική διαλογή με αύξουσα σειρά στη λίστα με τους κόμβους.

Τελικά αρχίζουμε την εισαγωγή των κόμβων στο γράφο. Στη πρώτη γραμμή του γράφου εισάγουμε του κόμβους και στην επόμενη τις τιμές που έχει ο κάθε κόμβος. Έτσι δημιουργούμε ένα πίνακα που έχει τη συγκεκριμένη δομή:

1	2	3	4	5	6	7	8
2	0	0	2	2	0	1	0
2	0	0	2	1	0	1	0

Παράδειγμα πρώτης διάσχισης του γράφου για το δεύτερο παράδειγμα της εργασίας με 3 εισόδους.

VCC=1 , GND=0 , X=2 , Z=3 ,SC=4

Η συνάρτηση colorGraph() έλεγχος για βραχυκύκλωμα:

```
//Color the next node of the graph
for (int i = 0; i < netlistRows; i++)
{
    mosType = netlist[i][0];
    gate = netlist[i][1];
    node0 = netlist[i][2];
    node1 = netlist[i][3];

    //Check for short circuits.
    if ((mosType==0) && (graph[currentGraph][findGraphPos(node0)]==1) && (graph[currentGraph][findGraphPos(node1)]==0) && (graph[currentGraph][findGraphPos(gate)]==0))
    {
        graph[currentGraph][findGraphPos(node0)]=4;
        //This is Short Circuit
        shortCircuit = true;
        break;
    }
    if ((mosType==0) && (graph[currentGraph][findGraphPos(node1)]==1) && (graph[currentGraph][findGraphPos(node0)]==0) && (graph[currentGraph][findGraphPos(gate)]==0))
    {
        graph[currentGraph][findGraphPos(node1)]=4;
        //This is Short Circuit
        shortCircuit = true;
        break;
    }
    if ((mosType==1) && (graph[currentGraph][findGraphPos(node0)]==1) && (graph[currentGraph][findGraphPos(node1)]==0) && (graph[currentGraph][findGraphPos(gate)]==1))
    {
        graph[currentGraph][findGraphPos(node0)]=4;
        //This is Short Circuit
        shortCircuit = true;
        break;
    }
    if ((mosType==1) && (graph[currentGraph][findGraphPos(node1)]==1) && (graph[currentGraph][findGraphPos(node0)]==0) && (graph[currentGraph][findGraphPos(gate)]==1))
    {
        graph[currentGraph][findGraphPos(node1)]=4;
        //This is Short Circuit
        shortCircuit = true;
        break;
    }
```

```
//check if short circuit was found
if (shortCircuit)
{
    printf("\n\nERROR: The circuit has a shortcircuit. Please try another!\n\n");
    printf("ShortCircuit in NODES %d - %d\n",node0,node1);
    printf("Press ENTER key to continue");
    fprintf(fpOut,"\n\nERROR: The circuit has a shortcircuit. Please try another!\n\n");
    fprintf(fpOut,"ShortCircuit in NODES %d - %d\n",node0,node1);
    getchar();
    break;
}
```

Σε αυτό το σημείο ο αλγόριθμος τρέχει μέχρι να συγκλίνει ο γράφος ή να βρούμε κάποιο βραχυκύκλωμα.

Αρχικά διαβάζουμε το netlist και αποθηκεύουμε τα στοιχεία του CMOS που μας ενδιαφέρουν όπως το Source και το Drain καθώς και το αν είναι PMOS ή NMOS και την τιμή του GATE.

Τρέχουμε τη διαδικασία για κάθε γραμμή του netlist μέχρι να κάνουμε μια αλλαγή στο γράφο. Δηλαδή το netlist σε κάθε επανάληψη χρωματισμού του γράφου το διαβάζουμε από πάνω προς τα κάτω και προχωράμε στην επόμενη γραμμή μέχρι να βρούμε ένα σημείο που θα χρωματίσουμε τον επόμενο κόμβο του γράφου. Αφού κάνουμε το χρωματισμό δεν κάνουμε άλλες αλλαγές στον γράφο και προχωράμε ώστε να κάνουμε τη διαδικασία ξανά από την αρχή.

Ελέγχουμε αν υπάρχει ένα βραχυκύκλωμα κοιτάζοντας αν ο κόμβος που πάμε να πειράξουμε στο netlist είναι 1 και προσπαθούμε να του δώσουμε την τιμή 0. Όταν γίνει αυτό βάζουμε στον κόμβο την τιμή 4 και σταματάμε τη διαδικασία χρωματισμού του γράφου. Ενημερώνουμε τη μεταβλητή shortCircuit , βγαίνουμε από τον βρόγχο και τερματίζει η διαδικασία για το συγκεκριμένο διάνυσμα εισόδου με το κατάλληλο μήνυμα λάθους ώστε να αναγνωρίσει ο χρήστης που συμβαίνει το βραχυκύκλωμα .

Η συνάρτηση colorGraph() για χρωματισμό του επόμενου κόμβου:

```
//Color next node
if ((mosType==0) && (graph[currentGraph][findGraphPos(node0)]==2) && (graph[currentGraph][findGraphPos(node1)]!=2) && (graph[currentGraph][findGraphPos(gate)]==0))
{
    graph[currentGraph][findGraphPos(node0)]=graph[currentGraph][findGraphPos(node1)];
    break;
}
//Color next node
if ((mosType==0) && (graph[currentGraph][findGraphPos(node1)]==2) && (graph[currentGraph][findGraphPos(node0)]!=2) && (graph[currentGraph][findGraphPos(gate)]==0))
{
    graph[currentGraph][findGraphPos(node1)]=graph[currentGraph][findGraphPos(node0)];
    break;
}
//Color next node
if ((mosType==1) && (graph[currentGraph][findGraphPos(node0)]==2) && (graph[currentGraph][findGraphPos(node1)]!=2) && (graph[currentGraph][findGraphPos(gate)]==1))
{
    graph[currentGraph][findGraphPos(node0)]=graph[currentGraph][findGraphPos(node1)];
    break;
}
//Color next node
if ((mosType==1) && (graph[currentGraph][findGraphPos(node1)]==2) && (graph[currentGraph][findGraphPos(node0)]!=2) && (graph[currentGraph][findGraphPos(gate)]==1))
{
    graph[currentGraph][findGraphPos(node1)]=graph[currentGraph][findGraphPos(node0)];
    break;
}
```

Στο σημείο αυτό χρωματίζουμε τον επόμενο βρόγχο. Ελέγχουμε αν ο κόμβος που προσπαθούμε να χρωματίσουμε είναι X και αν είναι βάζουμε την κατάλληλη τιμή σύμφωνα με το Netlist.

Η συνάρτηση colorGraph() για έλεγχο σύγκλισης του γράφου:

```
//check if graph converged
//#########################
for (int i = 0; i < numOfNodes; i++)
{
    if (graph[currentGraph-1][i]==graph[currentGraph-2][i])
    {
        sameAsPrev++;
    }
}
if (sameAsPrev==numOfNodes)
{
    printf("\nGraph converged\n\n");
    fprintf(fpOut,"\nGraph converged\n\n");

    converged =true;
    for (int i = 0; i < numOfTestOut; i++)
        {
            printf("Output Node %d = %d\n\n",testOut[i],(graph[currentGraph-1][findGraphPos(testOut[i])]));
            fprintf(fpOut,"Output Node %d = %d\n\n",testOut[i],(graph[currentGraph-1][findGraphPos(testOut[i])]));
            if ((graph[currentGraph-1][findGraphPos(testOut[i])])==2)
            {
                printf("ERROR: Node %d is Z! This is an open Circuit! Please fix this and try again!\n");
                fprintf(fpOut,"ERROR: Node %d is Z! This is an open Circuit! Please fix this and try again!\n");
                correctSims--;
            }
        }
        correctSims++;
    printf("Press ENTER key to continue");
    getchar();
}
//#########################
//End check
```

Σε αυτό το σημείο ελέγχουμε τις τελευταίες 2 εκδόσεις του γράφου και αν είναι ίδιες σταματάμε αφού ο γράφος μας έχει συγκλίνει.

Ελέγχουμε αν η έξοδος έχει τιμή 0 ή 1 και αν έχει τιμή 2 δηλαδή X εκτυπώνουμε μήνυμα λάθους και δεν αυξάνουμε το μετρητή που μας μετράει τις σωστές προσομοιώσεις του κυκλώματος.

Κάθε φορά που τελειώνει και μια προσομοίωση για ένα διάνυσμα τιμών του κυκλώματος το πρόγραμμα σταματάει ώστε να μπορούμε να δούμε πιο εύκολα πώς πήγε η διαδικασία και τυχόν λάθη. Για να προχωρήσουμε αρκεί να πατήσουμε το κουμπί Enter στο πληκτρολόγιο μας. Χρησιμοποιήθηκε η συνάρτηση getchar() για αυτή τη διαδικασία έναντι της συνάρτησης getch() γιατί η δεύτερη δεν ανήκει στην standardC και πιθανόν να δημιουργούσε προβλήματα σε συστήματα LINUX.

Η συνάρτηση findGraphPos():

```
//Find the position of a node
int findGraphPos(int nodeVal)
{
    for (int i = 0; i < numOfNodes; i++)
    {
        if (graph[0][i]==nodeVal)
            {
                return i;
            }
    }
    return -1;
}
```

Η συνάρτηση διαβάζει ένα κόμβο δηλαδή το νούμερο που έχουμε αντιστοιχήσει μέσα στο αρχείο εισόδου και επιστρέφει τη θέση του στον πίνακα του γράφου.

Η συνάρτηση printGraph():

```
void printGraph(FILE* fpOut)
{   //Print graph
        //#########################
        for (int i = 0; i < numOfGraphs-1; i++)
        {

            printf("\nGraph%d: ",i);
            fprintf(fpOut,"\nGraph%d: ",i);
            for (int j = 0; j < numOfNodes; j++)
            {
                printf(" %d",graph[i][j]);
                fprintf(fpOut," %d",graph[i][j]);
            }
        }
        //#########################
}
```

Είναι υπεύθυνη για την εκτύπωση του γράφου

Ανάλυση της Διαδικασίας

Αφού διαβάσουμε τα δεδομένα μας ο χρωματισμός του γράφου γίνεται με τη παρακάτω διαδικασία.

Παράδειγμα με χρήση του κυκλώματος που δόθηκε για 3 εισόδους.

NETLIST:

0	3	7	5
0	8	5	4
0	6	5	4
1	8	4	1
1	6	1	2
1	3	4	2

Γράφοι που δημιουργήθηκαν για διάνυσμα εισόδου <0,0,0>:

Κόμβοι	1	2	3	4	5	6	7	8
1η Επανάληψη	2	0	0	2	2	0	1	0
2η Επανάληψη	2	0	0	2	1	0	1	0
3η Επανάληψη	2	0	0	1	1	0	1	0
4η Επανάληψη	2	0	0	1	1	0	1	0

Διαβάζουμε το Netlist.
Η πρώτη αλλαγή που θέλουμε να κάνουμε είναι να δώσουμε στον κόμβο 5 τη τιμή του κόμβου 7 , όπως ορίζεται από την πρώτη γραμμή του netlist.

Σε κάθε επανάληψη αλλάζουμε τιμή σε ένα μόνο κόμβο.

Η δεύτερη αλλαγή που θέλουμε να κάνουμε είναι να δώσουμε στον κόμβο 4 τη τιμή του κόμβου 5, όπως ορίζεται στην δεύτερη γραμμή του netlist, αφού πλέον γνωρίζουμε τη τιμή του.

Στην επόμενη επανάληψη δεν έχουμε αλλαγές που να μπορούμε να κάνουμε σύμφωνα με το netlist.

Βλέπουμε ότι η 3η με τη 4η επανάληψη είναι ίδιες και σταματάμε την διαδικασία.

Δεν βρήκαμε κάποια επανάληψη που να συμβαίνει βραχυκύκλωμα.

Διαβάζουμε την τιμή που έχει η έξοδος του κυκλώματος αν αυτή είναι 2 δηλαδή X ενημερώνουμε το χρήστη για ανοικτό κύκλωμα στην έξοδο.

Αφού δεν συμβαίνει εκτυπώνουμε την τιμή της εξόδου.

Παράδειγμα εκτύπωσης του προγράμματος για το παραπάνω παράδειγμα

```
RAILS ARE :
VCC = 7
GND = 2
INPUTS ARE :
3 8 6
OUTPUTS ARE :
4
NETLIST IS : (where 0 in first column = PMOS and 1 = NMOS)
0 3 7 5
0 8 5 4
0 6 5 4
1 8 4 1
1 6 1 2
1 3 4 2
TEST IN IS :
3 6 8
TEST OUT IS :
4
TEST VECTORS ARE :
Input:3 = 0 Input:6 = 0 Input:8 = 0
SIMULATING CIRCUIT...

Number of Nodes: 8
Nodes :
1 2 3 4 5 6 7 8
Graph0:  1 2 3 4 5 6 7 8
Graph1:  2 0 0 2 2 0 1 0

Repetition 1 Graph Values are: VCC=1 , GND=0 , X=2 , Z=3 , SC=4

Graph0:  1 2 3 4 5 6 7 8
Graph1:  2 0 0 2 2 0 1 0
Graph2:  2 0 0 2 1 0 1 0

Repetition 2 Graph Values are: VCC=1 , GND=0 , X=2 , Z=3 , SC=4

Graph0:  1 2 3 4 5 6 7 8
Graph1:  2 0 0 2 2 0 1 0
Graph2:  2 0 0 2 1 0 1 0
Graph3:  2 0 0 1 1 0 1 0

Repetition 3 Graph Values are: VCC=1 , GND=0 , X=2 , Z=3 , SC=4

Graph0:  1 2 3 4 5 6 7 8
Graph1:  2 0 0 2 2 0 1 0
Graph2:  2 0 0 2 1 0 1 0
Graph3:  2 0 0 1 1 0 1 0
Graph4:  2 0 0 1 1 0 1 0
Graph converged

Output Node 4 = 1

Press ENTER key to continue
```

Συμπέρασμα

Το πρόγραμμα λειτουργεί κανονικά και εμφανίζει όλα τα πιθανά προβλήματα που μπορούμε να συναντήσουμε σε προσομοίωση απλών κυκλωμάτων. Δίνει στο χρήστη αρκετές πληροφορίες για να γνωρίζει αν το κύκλωμα που έδωσε είναι σωστό και το σημείο που συμβαίνει κάποιο πρόβλημα ώστε να μπορεί ευκολότερα να το φτιάξει.